Journal: New Phytologist Article type: Modeling /Theory

OpenSimRoot: Widening the scope and application of root architectural models

Postma, Johannes A.[1], Kuppe, Christian[1] , Owen, Markus R.[2,3], Mellor, Nathan[3,4], Griffiths, Marcus[3,4], Bennett, Malcolm J.[3,4], Lynch Jonathan P.[3,4,5], Watt, Michelle [1]

1) Plant Sciences, Institute of Bio and Geosciences 2, Forschungszentrum Jülich, Wilhelm-Johnen Straße 52425 Jülich, Germany

2) Centre for Mathematical Medicine and Biology, School of Mathematical Sciences, University of Nottingham, Nottingham, NG7 2RD, UK

3) Centre for Plant Integrative Biology, University of Nottingham, Nottingham, LE12 5RD, UK

4) Plant & Crop Sciences Division, School of Biosciences, University of Nottingham, Nottingham, LE12 5RD, UK

5) Department of Plant Science, Pennsylvania State University, 102 Tyson Building, University Park, PA 16802, USA

Author for correspondence: Johannes A. Postma, email: j.postma@fz-juelich.de, telephone: +49(0)2461614333, twitter: @j_a_postma

Total word count: (regular research papers should not exceed 6500 words): Currently 6254

Word count Summary: 193

Word count Introduction: 669

Word count Description 3866

Word count Results 1075

Word count Discussion 451

Word count Acknowledgements & Author contributions: 217

Number of Figures: 6

Number of Tables: 1

Summary

- **Research Conducted and Rationale:** *OpenSimRoot* is an open sourced, functional-structural plant model and mathematical description of root growth and function. We describe *OpenSimRoot* and its functionality to broaden the benefits of root modeling to the plant science community.
- **Description:** *OpenSimRoot* is an extended version of *SimRoot*, established to simulate root system architecture, nutrient acquisition, and plant growth. *OpenSimRoot* has a plugin, modular infrastructure, coupling single plant and crop stands to soil nutrient, and water transport models. It estimates the value of root traits for water and nutrient acquisition in environments and plant species.
- **Key results and unique features:** The flexible *OpenSimRoot* design allows upscaling from root anatomy to plant community to estimate 1) resource costs of developmental and anatomical traits, 2) trait synergisms, 3) (inter species) root competition. OpenSimRoot can model 3D images from MRI and X-ray CT of roots in soil. New modules include: 1) soil water dependent water uptake and xylem flow, 2) tiller formation, 3) evapotranspiration, 4) simultaneous simulation of mobile solutes, 5) mesh refinement, and 6) root growth plasticity.
- **Conclusion:** *OpenSimRoot* integrates plant phenotypic data with environmental metadata to support experimental designs and gain mechanistic understanding at system scales.

Keywords: Root system architecture, Functional Structural Plant Model, OpenSimRoot, Root architectural traits, Simulation, Model driven Phenotyping, Plant nutrition

Introduction

Functional-structural plant models combine a representation of 3D plant structure with physiological functions to advance plant science and its applications (Vos *et al.*, 2010; Dunbabin *et al.*, 2013). Those that incorporate below-ground root parameters (Dunbabin *et al.*, 2002; Pagès *et al.*, 2004; Wu *et al.*, 2007; Pierret *et al.*, 2007; Javaux *et al.*, 2008; Leitner *et al.*, 2010; Lobet *et al.*, 2014; Gérard *et al.*, 2017), require significant time and expertise in biological, mathematical, computational and digital image analyses, and therefore their development benefits greatly from an open and global setting. *SimRoot* is one of the most feature-rich and highly cited functional-structural root architectural models. However the last full description dates back twenty years (Lynch *et al.*, 1997), and subsequent papers report application of the model, with successive changes embedded in methods sections (Postma & Lynch, 2011a,b; Dathe *et al.*, 2013). Here we describe fully a new, open source version, branded *OpenSimRoot*, that is freely available for download (http://rootmodels.gitlab.io/*OpenSimRoot*). New features in this version allow simulation of more growth scenarios and crops, and its application has been widened to support emerging root phenotyping technologies.

SimRoot was originally designed to reconstruct root system architecture (RSA, see Table 1) from empirical data such as growth rates, angles and branching frequencies of different root classes. A post-simulation analysis of root geometry, nutrient uptake, and carbon costs enabled comparison of different RSAs with respect to their efficiency in taking up phosphorus relative to carbon costs (Nielsen *et al.*, 1994; Lynch & Beebe, 1995; Nielsen *et al.*, 1997; Lynch *et al.*, 1997; Ge *et al.*, 2000; Rubio *et al.*, 2001; Walk *et al.*, 2004, 2006). Later versions coupled physiological mechanisms such as root respiration, nutrient uptake, canopy photosynthesis, and RSA to simulate how the root phenotype dynamically interacts with the soil environment, and how this interaction influences acquisition of soil resources and consequently plant growth (Postma & Lynch, 2011a,b, 2012; Dathe *et al.*, 2013; Postma *et al.*, 2014a; Dathe *et al.*, 2016; York *et al.*, 2016). The initial focus was on phosphorus capture (Lynch & Beebe, 1995; Ge *et al.*, 2000; Ma *et al.*, 2001; Postma & Lynch, 2011b), which was later expanded to include C (photosynthesis), N, K, and water (Postma *et al.*, 2008; Postma & Lynch, 2011a; Dathe *et al.*, 2013). Microeconomic theory, in which resource acquisition is compared to resource investment costs, has guided the interpretation of results (Lynch, 2007; Postma *et al.*, 2014b). Although *SimRoot* was designed as a heuristic model, i.e., a tool for exploring implications of existing knowledge, and gaps in that knowledge, it proved surprisingly accurate for predicting fitness outcomes of root phenotypes (Chen *et al.*, 2011; Saengwilai *et al.*, 2014; Zhan *et al.*, 2015).

SimRoot is one of several root models that have been developed. Dunbabin et al. (2013) presents an exhaustive review of all root models to date and their capabilities. To our knowledge *OpenSimRoot* is currently the only plant root model that is openly version controlled (GIT) and GPLv3 licensed, allowing community-driven development. We envision that *OpenSimRoot* will be used and expanded by both modelers and non-modelers to simulate RSA and nutrient and water uptake in an ever widening scenario for species, environments and crop management practices to advance root-based opportunities to increase resource-efficient agricultural productivity. A design goal of *OpenSimRoot* is a flexible model structure that can be controlled by the user rather than the programmer. This means that, through a plugin infrastructure, the user can directly vary components of the model and compare the results. Model behavior can further be studied through sensitivity analysis, which has been a major focus in past publications.

In this paper we initially provide a short description of the design of the *OpenSimRoot* model and definitions, then present the major submodels in *OpenSimRoot* which simulate RSA, the shoot, carbon, water and nutrient acquisition and utilization, root growth plasticity, and geometric descriptors. After this model description we discuss model implementation, which is designed for flexibility, extensibility, transparency and robust numerics. We conclude with several examples of *OpenSimRoot* usage.

Model description

OpenSimRoot has, compared to other root models, a unique design which centers on coupling various mini-models (For definitions see Table 1). The distinction between parameter and algorithm has been, in line with object oriented programming, removed by encapsulating both within classes which share a common interface for coupling and data exchange.

OpenSimRoot design

OpenSimRoot contains a command line interface (CLI), a simulation engine, a plugin library, and classes responsible for reading and writing of data (Figure 1, Notes S1,2,3&4). The simulation engine implements an application programming interface (API) through which different modules can request information (See Note S1). The plugin infrastructure allows developers to implement new modules with limited knowledge about the rest of the code. Each plugin establishes dependencies between minimodels through the API and requests data of other minimodels in order to compute the necessary information. At the start of execution the import module reads an XML file (see below) and, based on that file, constructs a tree of minimodels. According to the specification in the XML, the minimodels load (instantiate) appropriate algorithms from a registry which lists all available plugins (Note S3). The plugin infrastructure not only allows the user to implement new processes, but also to implement alternative algorithms and compare model results. The behavior of the modules described below is thus not fixed but can be adapted to hypotheses. Simulation is driven based on data requests that originate from the users request for output. Upon instantiation of the object tree, the modules that write output start requesting information in order to write the output files. The CLI has a small number of options (listed with –h) with the most important one the input file name. Runs are non-interactive such that many runs with different parameter combinations can be fully automated on a computational cluster. This capability is important when large numbers of simulations are required, for example when exploring parameter sensitivity or processing real root structures (see Results) from large numbers of plants.

Description of the various modules

Root growth and RSA. The root system is represented by vertices and edges in *OpenSimRoot*. Every root tip has its own vertex with dynamic coordinates, and all other vertices have stationary coordinates that are placed behind the root tip as it extends. The final discretization of the root system can be coarser than the frequency of each growth point's directional change. A fine scale discretization request can automatically reduce the integration time step. In the case of a coarse discretization, the length of a root segment is not the linear distance between two vertices, but the true distance that the root grew, based on growth rate at that given time. We thus "simplify" the growth trajectory for computational reasons, without losing the true root length.

To grow a root system, we need: 1) when and where root tips (primordia) are created, 2) how fast root tips grow, and 3) in what direction. To start, we assume that, at a minimum, one primary root and a hypocotyl are present in the seed embryo. The term hypocotyl is used here freely to include any shoot axiles (stems) that are the origins of adventitious roots, whether simulating dicotyledons or monocotyledons. Branch roots and their own branch roots (classed according to order), are assumed to emerge from the primary root, based on rules that control the timing and placing of the branches. Adventitious roots (crown or nodal roots in grasses) can branch from the stem according to different schemes; the simplest defined by a starting time and position of a single whorl of roots. Formation of branch roots from these axiles is typically based on branching frequencies, which can be expressed in time, or space, or both, where the missing information is computed based on the growth rate of the parent root. Roots can branch from either phloem or xylem poles, depending on species (Casimiro *et al.*, 2003). The number of poles determines number of positions of the radial branching angle, while the axial branching angle (angle between parent root and branch) is given in the parameter section (for detailed explanation see Lynch, 1997, and Figure 2).

Elongation rates of individual roots are predefined in the parameter space, but might be scaled according to a “root vigor” scaling factor; for example drawn from a lognormal distribution of elongation rates (scaled to unity), thus creating variation in length. The vigor factor can also scale the root specific root diameter, to allow an allometric relation between elongation rate and root diameter expansion within a root class (Pagès, 2000; Wu *et al.*, 2016). Initial root diameter is otherwise a root class specific input parameter.

While initial growth direction is set by specified radial and axial branching angles (Figure 2), the direction can be changed with a tropism vector. The tropism vector is the sum of several vectors representing gravitropism, random impedance, and nutrient tropisms and is added to the normalized growth direction vector, to obtain the new direction.

Once the root is growing, its branching rules allow it to branch off new roots of different classes and the whole process is repeated. Although *OpenSimRoot* currently does not simulate shoot architecture, a simple tiller model is included. Tillers can form their own leaf area, and their own root systems. In grasses, tillers produce nodal roots which can form a significant fraction of the total root system, depending on species and environment (Atkinson *et al.*, 2014; Sebastian *et al.*, 2016). Tiller formation is done on the basis of a table that indicates the time dependent delay till the next tiller is formed. Dicotolydonous roots have secondary growth from cambia which thicken the stele and periderm in the root. Secondary growth is simulated using a time dependent radial growth rate, scaled to distance along the root.

Simulation of shoot growth and related processes. A simple shoot model can be constructed with *OpenSimRoot* plugins. The shoot model is non-geometric and represents the shoot by the state variables leaf area and leaf and stem dry weight. Increase in dry weights is based on carbon allocated to leaves and stems, multiplied by a dry weight to carbon factor. Increase in leaf area is the increase in leaf dry weight multiplied by the specific leaf area (SLA). Carbon partitioning can be based on predefined time dependent values (van Ittersum *et al.*, 2003). Carbon partitioning tables are typically established from dry weight measurements, and thus instead of entering carbon partitioning tables, *OpenSimRoot* can also compute partitioning directly from dry weight measurements. This predefined growth represents “potential” growth under a well-watered and fertilized condition, whereas nutrient or carbon limitations may alter carbon partitioning (see below). Total carbon available for plant growth is computed by subtracting the carbon costs (for example respiration, and root exudates) from the total carbon fixed in the leaves, and or available from seed or non-structural carbon reserves. Carbon costs depend on rates of respiration or carbon expenditure on exudates or nitrate uptake, and these are integrated over the whole plant or root system. Total carbon fixation is based on a radiation use efficiency (RUE) model, whereby intercepted light is converted linearly to carbon fixation. Intercepted light is computed from leaf area index, assuming that the simulated plant is in a homogeneous canopy of equally spaced and identical plants. Tillers are simulated as new plants with their own leaf area, but sharing resources.

Carbon allocation to roots. The root growth module can compute the carbon for growth for each root segment (edge) using its volumetric increase, and a specific root volume (g cm^{-3}). Volume increases arise from primary and/or secondary growth, and root segments are assumed cylindrical or, in the case of varying diameters, a truncated cone. *OpenSimRoot* compares available to required carbon, and if source strength is greater than sink strength, stores the carbon left over into a labile pool. *OpenSimRoot* thus considers that plant growth may be physiologically, not resource, constrained (Postma *et al.*, 2014b). The labile pool is depleted when sink strength (defined by carbon needed for potential growth) is greater than source strength. Once stored carbon is depleted, growth rates decline. Various rules for carbon allocation under source limiting conditions have been implemented. The most used rule to date prioritizes shoot over roots, and within the root system, secondary growth (root cambial thickening) over elongation, and within the root classes, elongation of major bearing roots over branch roots. Consequently, when plant growth is carbon limited, growth rates of branch roots are reduced more than the growth of the parent roots. These rules do not have a physiological basis, rather a pragmatic basis in which source sink imbalances are seen as errors in the parameterization and estimation of the growth rates, and the assumption is that these errors are more likely in the branch root growth than in the shoot growth. However, other rules, such as equal scaling of all organs have been implemented, and can be used if the user assumes that all sinks compete equally for the available carbon.

Although the inputs of the model are absolute growth rates, allometric scaling, based on the ratio between actual and potential leaf area (not mass), can reduce the attainable growth rate of the canopy and the rate of formation of new root branches. This implies that plants can never fully recover from a stress. However, a recovery rate can be defined which allows the plant to grow, for example, 10% faster when resources permit. Allometric scaling can also be used for the formation of branches. For example, the number of nodal roots per whorl in maize is dependent on the size of the shoot.

Hydrology. *OpenSimRoot* includes a hydrology module (Figure 3). The implementation of the hydrology module involves the coupling of three models that simulate the movement of water through soil, plant and into the atmosphere. *OpenSimRoot* includes a simplified C++ implementation of the SWMS model which is used to simulate soil water transport in Hydrus (Diamantopoulos *et al.*, 2013) and RSWMS (Šimunek *et al.*, 1995). Water transport through the xylem is simulated using a hydraulic network model (Alm *et al.*, 1992; Doussan *et al.*, 1998) and evapotranspiration is simulated using the Penman-Monteith equation (Penman, 1948; Monteith, 1964). Small adjustments of these models, to achieve good coupling, are described in Note S5.

The hydrology module provides 3D water uptake profiles, drives convective nutrient transport, and can simulate compensatory water uptake and hydraulic redistribution, which may occur when the top soil dries out, causing nutrient uptake from dry soil domains to be reduced. It currently does not simulate drought related growth responses.

Nutrients. *OpenSimRoot* has a nutrient module to simulate the simultaneous uptake of solutes, originally implemented to simulate the impact of RSA on nutrient uptake, and to test tradeoffs for acquisition of nutrients (Postma & Lynch, 2011a; Dathe *et al.*, 2013). Postma et al. (2014a) showed how the optimal branching density in maize depends on the relative availability of phosphorus and nitrogen. The module involves three parts: 1) simulation of plant nutrient requirements, 2) simulation of nutrient acquisition, and 3) stressors which define how suboptimal plant nutrient concentrations affect physiology or growth (Figure 4). Nutrients are simulated independently of each other, except that in step (3) the impact of suboptimal nutrient concentrations on a given state variable is aggregated using a maximum or averaging function. For example, nitrogen might affect photosynthesis more than phosphorus, but phosphorus might affect the leaf area expansion rate more strongly (see Dathe et al., 2013).

The nutrient requirements of the plant are determined by integrating over the whole plant biomass predefined optimal and minimal nutrient concentrations. The plant acquires nutrients through seed reserves, uptake by the root system, and optional nitrogen fixation. Uptake of nutrients by the root system is simulated by Michaelis-Menten kinetics, where movement of nutrients in the soil towards roots is simulated through convection-dispersion-diffusion equations. *OpenSimRoot* includes two different implementations for solving these equations: 1) The Barber-Cushman model (Itoh & Barber, 1983), which simulates depletion zones around individual root segments at high resolution, and is suitable for immobile nutrients like phosphorus; and 2) a reimplementation of the solute model included in SWMS3D (Šimunek *et al.*, 1995), which couples to the soil water model within the hydrology module (above), simulates the whole soil domain and is suitable for mobile nutrients like nitrate. More detailed descriptions of these models are given in Note S5.

When acquisition falls short of what is required, plant stress is assumed. Stress impact functions can be defined for components such as leaf expansion rate, photosynthesis rates, respiration rates, and root elongation rates or secondary growth. By making the initial response of the shoot stronger than that of the roots, the plant decreases shoot to root ratios when nutrient deficient (Postma & Lynch, 2011a). *OpenSimRoot* will move towards a functional equilibrium, although, due to the inherent slow nature of growth, and the relatively fast dynamics of other processes, this functional equilibrium might not be reached (Postma & Lynch, 2011b; Postma *et al.*, 2014b). The current implementation assumes that, internally, reallocation of nutrients is fast and perfect, such that all organs experience equal stress. This might be true for a nutrient like nitrogen, which typically causes chlorosis everywhere in the shoot, but might not be correct for other nutrients. The importance of simulation of nutrient redistribution in the plant still needs study, and would require implementation of a shoot architectural model in which the age and position of individual leaves or canopy strata are simulated.

Mineralization and rhizosphere processes. *OpenSimRoot* implements the Yang and Janssen model for mineralization (Yang & Janssen, 2000). This model assumes exponential decline of a carbon pool, via aging and decline in break down rate. Based on C/N ratios of the substrate and C/N ratios of the microbial biomass, the net mineralization or immobilization of N can be computed. *OpenSimRoot* assumes that ammonium is readily converted to nitrate, and soil water content and temperature are currently ignored. The implementation of the Yang and Janssen model in *OpenSimRoot* simulates mineralization for every FEM node independently and thus mineralization rates may vary in space. The user can define a nitrogen fixation rate as a percentage of the nitrogen requirements of the plant. Fixation will not directly reduce nitrogen uptake from soil, but improves plant nitrogen status.

Root exudation is not explicitly simulated, but is instead described as a root class- and time-dependent carbon cost. Furthermore, exudation may increase the soluble nutrient concentration in the soil at the cost of the insoluble fraction and thereby increase nutrient availability locally (Barber-Cushman model only).

Root growth plasticity. *OpenSimRoot* can define reaction curves to local environmental factors, to simulate a localized growth behavior of roots (Figure 5), often termed “plasticity” (Bradshaw, 1965; Palmer *et al.*, 2001). 3D interpolation of available environmental data is used to define values at the root surface. For example, a reaction curve (norm, (Pigliucci *et al.*, 1996)) could describe how gravitropism is scaled according to the local concentration of a nutrient. Similarly, branching frequency or root elongation rates can be scaled according to a local soil variable. For example, static fields for soil compaction can be defined in three dimensions, using lists of coordinates and associated values in conjunction with a spatial interpolation algorithm. Root elongation can then be defined as a function of local soil compaction.

Currently, only absolute values (scalars) of local environmental variables such as soil compaction or nutrient concentrations can be used to simulate plasticity responses. Gradient sensing (i.e. relative values or tensors) of environmental factors may be important for nutrient- or hydro-tropism, or root proliferation responses into enriched patches. However, the biological mechanism for sensing gradients is unclear, and currently no such mechanism has been implemented. *OpenSimRoot* does, however, include a mechanism to scale the strength of the local plasticity response on the basis of yet another reaction norm which might couple plasticity to whole plant status.

Root length distribution and virtual coring. *OpenSimRoot* can compute several geometric metrics, specifically root length density profiles, virtual coring, root length below D90 for nitrate, and overlap of depletion zones. Others, like explored soil volume, or fractal dimensions, can be computed by the user on the basis of the geometric model output.

Root anatomy. Root anatomy is not simulated in 3D explicitly, but *OpenSimRoot* can represent the stele diameter, thickness of the cortex, the degree of cortical senescence, the degree of root cortical aerenchyma formation, and the length, diameter and density of root hairs. These anatomical traits may influence processes at the root segment level, specifically nutrient content, respiration, nutrient uptake and hydraulic conductivity (Fan *et al.*, 2003; Hu *et al.*, 2014).

Implementation

OpenSimRoot is written in C++, an object oriented programming language. *OpenSimRoot* couples minimodels, which encapsulate the simulation of a single state variable. State variables are assumed to be associated with time and space and always have a unit. Minimodels are implemented as single C++ classes which inherit from the same base class (named SimulaBase), such that they all have the same interface (API). This interface allows minimodels to connect to other minimodels and request data. Minimodels might encapsulate a constant, an interpolation table, a random number generator or may make use of helper functions for computation. These helper functions are of the class type IntegrationBase and DerivativeBase and are registered under their specific names, such that, based on the input, the correct helper function can be instantiated. Helper functions compute a variable, and when associated with an integration function, can be integrated over time. The true functionality of *OpenSimRoot* is thus dispatched to the helper functions. Through a plugin framework, developers can add new helper functions and thus extend the functionality of the model. Example code for a plugin is given in Note S4.

Thus, coupling of the state variables is done through a simple common interface guaranteeing that minimodels are, from a programmer point of view, standalone objects. Computations are quite indifferent as to how dependent variables are computed. This creates high flexibility in the input files, where the state variables can be defined in a variety of ways, i.e. constant, stochastic, interpolation table, or based on a plugin (Table 2, Note S6).

One big challenge in coupling independent (mini)models is the implementation of numerical integration when different models have different time steps, and when implicit coupling is desired. In *OpenSimRoot* we implemented a general framework for predictor corrected methods, by default RungeKutta4, with three components: 1) Interpolation, 2) Prediction, and 3) Dependency tracking. Each minimodel keeps a time table to interpolate between time steps and return historical information. Different minimodels can run at different time steps, which are however synchronized at every globally defined maximum time step. Since all data requests loop through the SimulaBase API, *OpenSimRoot* tracks forward dependencies and predictions, to determine whether to keep the step taken. Interdependent minimodels (For an exemplar graph of dependencies see Note S7) update using a predictor corrector method with interpolation to ensure compatibility of time steps. Whilst the precise order may have some influence on numerical accuracy or efficiency, there is typically no rational basis on which to prefer any one order of evaluation and is therefore simply dependent on the order of information requests (Typically breadth-first search, see hierarchical contextualization).

The independent minimodel approach can create a significant computational overhead. However, simulations of RSA are still relatively fast compared to soil and we regard the ease with which new functionality can be added with no or little programming effort or knowledge about the rest of the code as more important than runtime.

The current implementation of *OpenSimRoot* only depends on the standard C++ libraries (ISO C++11, and a few system libraries for the CLI), and on our website (rootmodels.gitlab.io) we provide directions for compilation and running on Linux, Mac and Windows operating systems.

Hierachical contextualization. Many dynamic models are structured along a sequence of events; the 'time loop'. However, *OpenSimRoot* represents the plant as a hierarchy of interacting components to allow the main purpose of understanding of the function of root traits for the whole plant. Minimodels are placed in a simple hierarchy which provides them context, while the object oriented paradigm “hides” the internal workings of each component.

Dynamic adding of components. *OpenSimRoot* adds (instantiates) new components during simulation to represent newly grown roots. This contrasts with crop models that represent plant growth by an increase in values of the state variables. Dynamic memory management, connected to an object oriented programming paradigm, is a useful programming feature for adding new components (Dingkuhn *et al.*, 2005). Each minimodel can optionally have a class (inherited from the class ObjectGeneratorBase) attached to it, which, when the children of the minimodel are requested, is run to update the list of children. For example there are classes that will create new branch roots or will insert new vertices (rootNodes) into the hierarchy. Most of these classes do this by copying templates, which contain all the necessary minimodels that are defined in the input files. An example of an ObjectGenerator plugin is given in Note S4.

Input files

OpenSimRoot uses a hierarchical file of parameter values, which not only contains parameter values, but all state variables, and their metadata, such as names and units. Hierarchy provides context, such that parameter lists can be specific for different root classes of different plant species. Input files are implemented in XML, a general language for describing data together with metadata that is also hierarchical, flexible, allows comments, is supported by many software tools, and can be rendered in a browser as a more readable document. Note S6 gives an example of an input file that simulates a simple relative growth model.

OpenSimRoot allows the user not only to enter initial values, but arrays of initial time series. This way, part of the RSA can be predefined, based on measurements (also see examples in Results). This approach may be different from most models, but creates the opportunity to use the model as an extension to phenotyping as partial information derived from phenotypic measurements can be directly entered into the input files (Fiorani & Schurr, 2013, Figure 6). Parameterizations exist for maize, squash, bean, lupin, *Arabidopsis*, and barley, and are now being developed for wheat and rice (Ma *et al.*, 2001; Chen *et al.*, 2011; Postma & Lynch, 2012). Input files for maize and bean, a predefined root system, a small crop model and other testing scenarios are included in the source code repository (https://gitlab.com/rootmodels/OpenSimRoot).

Output files

OpenSimRoot includes export modules that can be enabled or disabled to retrieve specified output forms that include tables in text files, 3D models in various VTK (visual tool kit, www.vtk.org) formats, 3D raster images, and a XML formatted dump of the model in the format of *OpenSimRoot*'s own input files. For example: tables can be further processed with statistical software (like R), VTK files can be opened with 3D data viewers (e.g. Paraview, http://www.paraview.org/), and the model dump can be viewed in a web browser (Note S6).

License

OpenSimRoot is available under the GPLv3 Licence (https://www.gnu.org/licenses/gpl-3.0.en.html) which is an opensource – copyleft license. The license enables the practice of "good science" by making the model transparent and by facilitating contributions from a wider range of expertise in the community. Access the version controlled code at https://gitlab.com/rootmodels/OpenSimRoot.

Application examples for *OpenSimRoot*

SimRoot has found useful application in several domains, including 1) geometric analysis of root system form and function, 2) simulation of processes that are very difficult to measure empirically, 3) simulation of dynamic systems, 4) sensitivity analyses, and 5) simulation of hypothetical systems. In addition, a new capability of *OpenSimRoot* to read in (partially) predefined RSA enables application as an extension to 3D phenotyping techniques such as X-ray CT (Computed tomography) and MRI (Magnetic Resonance Imaging). Examples of all of these applications are provided below.

Studies on the function of RSA traits

A primary output of *OpenSimRoot* is the RSA phenotype emerging from input parameters simulating specific phenes like gravitropic setpoint angle or lateral root initiation interacting with environmental conditions. For example, due to spatio-temporal heterogeneity in soil nutrient availability, growth angles may differentially affect phosphorus and nitrogen uptake but also affect the degree of inter- versus intra-plant root competition (Ge *et al.*, 2000; Rubio *et al.*, 2001; Dathe *et al.*, 2013). Results of simulated maize-bean-squash intercropping systems showed that RSA and nitrogen fixation (bean) work towards reduced competition and increased biomass (Postma & Lynch, 2012; Zhang *et al.*, 2014). Competition among branches of the same parent root may become stronger when the root branching density increases, and since this increase results in greater sink strength, but not greater source strength (in carbon available for growth), the individual roots may stay shorter. Simulating these processes, Postma et al. (2014a) estimated that the optimal branching density (assuming parent roots have the same root branching density) for maize was lower when nitrogen availability decreased. The benefit of fewer but longer laterals in low nitrogen soils was confirmed in a genotypic contrast study (Zhan *et al.*, 2015). Walk et al. (Walk *et al.*, 2006) estimated the tradeoffs between basal root growth and adventitious root growth in bean and concluded that adventitious roots might be of most benefit when phosphorus availability is low. While these RSA traits represent tradeoffs, other traits may work in synergy towards greater productivity on low nutrient soils (Ma *et al.*, 2001; Postma & Lynch, 2011a; Miguel *et al.*, 2015). *OpenSimRoot* has also increased understanding of how integrated phenotypes function. This was demonstrated by York et al.(2015) who used *SimRoot* to estimate how changes in maize RSA, introduced by breeding over 100 years, might affect the nutrient uptake efficiency of modern cultivars. New functionality described here will enable new studies of the function of whole plant traits, such as tiller formation and its influence on RSA.

Relationships between RSA traits and root system descriptors.

Many researchers determine what might be called geometric descriptors of RSA: root length density profiles, fractal geometry, specific root length, total root length, rooting depth and convex-hull (Fitter & Stickland, 1992; Clark *et al.*, 2011). These descriptors can be computed on simulated roots and their relation to architectural, anatomical or functional traits can be inferred. For example, differences in the specific root length of a root system may be related to anatomical changes, or a different ratio of thick to finer roots. Nielsen et al. (1997) determined differences in fractal dimensions between phosphorus efficient and inefficient genotypes, and Walk et al. (2004) applied *SimRoot* to show how soil exploration for P related to the fractal dimensions of the root system. Miguel et al (2015) applied *SimRoot* to do “virtual coring” in order to support the idea that genotypic differences in rooting depth might best be seen when coring in between rows. These studies show how the geometric aspects of the root system can be related to root traits and function, something not easily derived from empirical measurements of actual root systems.

Scaling up from root anatomy to crop

At its smallest spatial scale, *OpenSimRoot* represents root anatomy, and at its largest scale it simulates crop measures like biomass, nutrient uptake and root zone depletion and leaching. For example, Ma et al., (2001) focused on root hairs in *Arabidopsis thaliana* and concluded that their length and density contribute synergistically towards greater phosphorus uptake. Chen et al., (2011, 2013) used *SimRoot* and lupin phenotypic data to compute that the contribution of root hairs to total phosphorus uptake might vary strongly among genotypes. Postma and Lynch (2011a,b) and Schneider et al. (unpublished) simulated the root class- and time-dependent formation of Root Cortical Aerenchyma (RCA) and Root Cortical Senescence (RCS) respectively, and determined that RCA and RCS may be mechanisms underlying greater growth on low nutrient soils in maize, bean and barley, possibly via efficient use and recycling of resources. Genotypic contrast studies on low N soils concur with these simulation results (Saengwilai *et al.*, 2014) which suggests that *OpenSimRoot* can be used for scaling up from anatomy to crop stands.

OpenSimRoot as an extension to plant phenotyping

Technologies like X-ray CT and MRI have been adapted to image root systems non-destructively and provide non-invasive ways to phenotype whole root systems in 3D in soil (Mooney *et al.*, 2012; Mairhofer *et al.*, 2013; van Dusschoten *et al.*, 2016). The utility to feed such data to a model was demonstrated by Stingaciu et al. (2013) for a non-growing lupin root system. Using time estimates, *OpenSimRoot* can simulate the growth of a root system such that the RSA is identical to that imaged. Figures 6a,b (for animation see Movie S1) show an MRI image, and the simulated root system. The simulation does not include a small portion (~8%) of the roots visible in the 3D image data because of limitations in image segmentation, rather than in the model. *OpenSimRoot* can add "MRI-non-visible" finer roots to the simulation according to existing model rules, and the simulation can be extended beyond the measured time, to predict continued growth of the root system. Importantly, *OpenSimRoot* modules for nutrient and water uptake can be enabled with the architectural phenotypes derived from measurements and simulation, and functions can be ascribed to the traits. This may help researchers and breeders go from image to functional understanding of the measured root systems, and compare genotypes not only on the basis of geometry, but also on the basis of modeled ability to take up water and nutrients. For example, Figures 6c,d show a CT image, and corresponding *OpenSimRoot* simulation of nitrate depletion zones around the root system. Integration of the model into phenotyping pipelines is also likely to help find deficits of the model, and give modelers a basis for improving parameterization and/or algorithms. This important development considerably widens the scope of application of *OpenSimRoot*.

Discussion & Conclusions

We have described the first open source version of the RSA model *SimRoot*, which is now available for use by biologists and modelers. New features that expand its use include hydrology to simulate and understand root system hydraulic properties. A novel area of application includes simulation of non-invasive 3D phenotypic data of RSA from MRI and X-ray CT, and their putative functions in nutrient and water uptake. To our knowledge, *OpenSimRoot* is currently the most feature rich and widely published multiplatform RSA model (Dunbabin *et al.*, 2013) that is freely available for direct download (http://rootmodels.gitlab.io/*OpenSimRoot*). The new open-source implementation combines features that will enable expansion of use for plant and crop science:

- a modular, plugin infrastructure for extending the model;
- a default predictor-corrected numerical scheme for integration and coupling;
- the ability to predefine any data that was measured, where the model will use the measured data instead of its algorithm for simulation (e.g. the root system, and optionally its history, may be partly pre-defined based on MRI or CT images);

- integration with a shoot model;
- ability to simulate competition among plants of different species;
- maintained by an international community of root researchers.

Relationships in crop models that are typically only defined empirically, such as competition among roots for nutrients, or root length density profiles, are actually a result of RSA, and therefore, RSA models provide insight into relations between measurable traits and emerging properties at the crop level. We regard the heuristic value of the model, and its use as a tool for developing and testing of concepts, and prediction of mechanisms and trends, as the more important motivation for model studies with, and continued development of, *OpenSimRoot*. The model may have further utility in extending phenotyping pipelines by estimating genotype performance based on measured root phenotypes.

Future development will be community driven, and may include new processes such as root signaling networks, drought responses, soil microbial interactions and soil chemistry. As our mechanistic understanding of different processes increases, *OpenSimRoot*'s hierarchical structure allows new empirical data to be represented by new algorithms. For example, gravitropism may be simulated on the basis of understanding of differential cell elongation rather than on the current empirically derived input. Open sourcing allows other modelers to couple *OpenSimRoot* to their models. For example shoot architectural models might be coupled to *OpenSimRoot*, in order to understand competition for light and shoot architectural traits in relation to RSA traits. Finally, opening up the code enables developers of other RSA models to compare the results of *OpenSimRoot* to those of their models, which may lead to constructive critique and improvements of all RSA models, and by extension, discoveries for improvements in understanding of plant and crop resource efficiencies.

Acknowledgements

We would like to acknowledge all the researchers who over the past 25 years have contributed to the development of *OpenSimRoot*, in particular RD Davis, Kai L. Nielsen, Gerardo Rubio, Zhenyang Ge, Raul Jaramillo, Tom Walk, Annette Dathe, Larry York, Eric Nord, Harini Rangajaran, Vera Hecht, Hannah Schneider and Ernst Schäfer. We would also like to thank Darren Wells and Dagmar van Dusschoten for contributing the CT and MRI images, respectively, and Daniel Pflugfelder for assistance in segmenting and analyzing the MRI image.

This research received support from the Forschungszentrum Jülich in the Helmholtz Association, and the German-Plant-Phenotyping Network that is funded by the German Federal Ministry of

Education and Research (project identification number: 031A053).

Author contributions

J.A.P. and M.W. planned the manuscript. J.P.L. conceived of SimRoot and led its development through 2011, J.A.P. rewrote the code, expanded its capabilities, and has led its development since 2011, with mathematical support from C.K. since 2013. J.A.P., C.K., N.M. and M.R.O. programmed various parts of the model code. All authors were involved in open sourcing of the code and forming a development team. M.J.B., N.M., M.G. and M.R.O. contributed the CT image data and the simulation output based upon that data. J.A.P., C.K., M.W., J.P.L. M.R.O. and M.J.B. wrote various parts of the manuscript, with input from all authors.

Supplemental files

Note S1: Description of the SimulaBase API

Note S2: How to run *OpenSimRoot*: description of the comman line interface CLI.

Note S3: Overview of all classes that form *OpenSimRoot*, including list of plugins

Note S4: Example C++ code for a plugin

Note S5: Technical description of water and nutrient modules

Note S6: Example input file

Note S7: Example graph of state variables and their dependencies

Movie S1: Animation of Figure 6.

References

Alm DM, Cavelier J, Nobel PS. **1992**. A finite-element model of radial and axial conductivities for individual roots: development and validation for two desert succulents. *Annals of Botany* **69**: 87–92.

Atkinson JA, Rasmussen A, Traini R, Voß U, Sturrock C, Mooney SJ, Wells DM, Bennett MJ. **2014**. Branching Out in Roots: Uncovering Form, Function, and Regulation. *Plant Physiology* **166**: 538–550.

Bradshaw AD. **1965**. Evolutionary significance of phenotypic plasticity in plants (EWC and JM Thoday, Ed.). *Advances in Genetics* **13**: 115–155.

Casimiro I, Beeckman T, Graham N, Bhalerao R, Zhang H, Casero P, Sandberg G, Bennett MJ. **2003**. Dissecting Arabidopsis lateral root development. *Trends in Plant Science* **8**: 165–171.

Chen YL, Dunbabin VM, Postma JA, Diggle AJ, Kadambot H. M. Siddique, Rengel Z. **2013**. Modelling root plasticity and response of narrow-leafed lupin to heterogeneous phosphorus supply. *Plant and Soil* **372**: 319–337.

Chen Y, Dunbabin V, Postma J, Diggle A, Palta J, Lynch J, Siddique K, Rengel Z. **2011**. Phenotypic variability and modelling of root structure of wild *Lupinus angustifolius* genotypes. *Plant and Soil* **348**: 345–364.

Clark RT, MacCurdy RB, Jung JK, Shaff JE, McCouch SR, Aneshansley DJ, Kochian LV. **2011**. Three-dimensional root phenotyping with a novel imaging and software platform. *Plant Physiology* **156**: 455–465.

Dathe A, Postma JA, Lynch JP. **2013**. Modeling resource interactions under multiple edaphic stresses. In: Timlin D, Ahuja LR, eds. Advances in Agricultural Systems Modeling. Enhancing Understanding and Quantification of Soil–Root Growth Interactions. Madison, Wis., USA: American Society of Agronomy, Crop Science Society of America, Soil Science Society of America. 273–294.

Dathe A, Postma JA, Postma-Blaauw MB, Lynch JP. **2016**. Impact of axial root growth angles on nitrogen acquisition in maize depends on environmental conditions. *Annals of Botany* **118**: 401–414.

Diamantopoulos E, Iden SC, Durner W. **2013**. Modeling non-equilibrium water flow in multistep outflow and multistep flux experiments. *HYDRUS Software Applications to Subsurface Flow and Contaminant Transport Problems*: 69-76.

Dingkuhn M, Luquet D, Quilot B, de Reffye P. **2005**. Environmental and genetic control of morphogenesis in crops: towards models simulating phenotypic plasticity. *Australian Journal of Agricultural Research* **56**: 1289–1302.

Doussan C, Pagès L, Vercambre G. **1998**. Modelling of the hydraulic architecture of root systems: An integrated approach to water absorption - Model description. *Annals of Botany* **81**: 213–223.

Dunbabin VM, Diggle AJ, Rengel Z, van Hugten R. **2002**. Modelling the interactions between water and nutrient uptake and root growth. *Plant and Soil* **239**: 19–38.

Dunbabin VM, Postma JA, Schnepf A, Loïc Pagès, Mathieu Javaux, Lianhai Wu, Daniel Leitner, Ying L. Chen, Zed Rengel, Art J. Diggle. **2013**. Modelling root–soil interactions using three–dimensional models of root growth, architecture and function. *Plant and Soil* **372**: 93–124.

van Dusschoten D, Metzner R, Kochs J, Postma JA, Pflugfelder D, Buehler J, Schurr U, Jahnke S. **2016**. Quantitative 3D analysis of plant roots growing in soil using magnetic resonance imaging. *Plant Physiology* **170**: 1176–1188.

Fan M, Zhu J, Richards C, Brown KM, Lynch JP. **2003**. Physiological roles for aerenchyma in phosphorus-stressed roots. *Functional Plant Biology* **30**: 493–506.

Fiorani F, Schurr U. **2013**. Future scenarios for plant phenotyping. *Annual Review of Plant Biology* **64**: 267–291.

Fitter AH, Stickland TR. **1992**. Fractal characterization of root system architecture. *Functional Ecology* **6**: 632–635.

Ge ZY, Rubio G, Lynch JP. **2000**. The importance of root gravitropism for inter-root competition and phosphorus acquisition efficiency: results from a geometric simulation model. *Plant and Soil* **218**: 159–171.

Gérard F, Blitz-Frayret C, Hinsinger P, Pagès L. **2017**. Modelling the interactions between root system architecture, root functions and reactive transport processes in soil. *Plant and Soil* **413**: 161–180

Hu B, Henry A, Brown KM, Lynch JP. **2014**. Root cortical aerenchyma inhibits radial nutrient transport in maize (Zea mays). *Annals of Botany* **113**: 181–189.

Itoh S, Barber SA. **1983**. A numerical solution of whole plant nutrient uptake for soil-root systems with root hairs. *Plant and Soil* **70**: 403–413.

van Ittersum MK, Leffelaar PA, van Keulen H, Kropff MJ, Bastiaans L, Goudriaan J. **2003**. On approaches and applications of the Wageningen crop models. *European Journal of Agronomy* **18**: 201–234.

Javaux M, Schroeder T, Vanderborght J, Vereecken H. **2008**. Use of a three-dimensional detailed modeling approach for predicting root water uptake. *Vadose Zone Journal* **7**: 1079–1088.

Leitner D, Klepsch S, Bodner G, Schnepf A. **2010**. A dynamic root system growth model based on L-Systems. *Plant and Soil* **332**: 177–192.

Lobet G, Pagès L, Draye X. **2014**. A modeling approach to determine the importance of dynamic regulation of plant hydraulic conductivities on the water uptake dynamics in the soil-plant-atmosphere system. *Ecological Modelling* **290**: 65–75.

Lynch J. 1995. Root Architecture and Plant Productivity. *Plant Physiology* **109**: 7–13

Lynch JP. **2007**. Rhizoeconomics: The roots of shoot growth limitations. *HortScience* **42**: 1107–1109.

Lynch JP, Beebe SE. **1995**. Adaptation of beans (*Phaseolus vulgaris* L.) to low phosphorus availability. *HortScience* **30**: 1165–1171.

Lynch JP, Nielsen KL, Davis RD, Jablokow AG. **1997**. *SimRoot*: Modelling and visualization of root systems. *Plant and Soil* **188**: 139–151.

Ma Z, Walk TC, Marcus A, Lynch JP. **2001**. Morphological synergism in root hair length, density, initiation and geometry for phosphorus acquisition in *Arabidopsis thaliana*: A modeling approach. *Plant and Soil* **236**: 221–235.

Mairhofer S, Zappala S, Tracy S, Sturrock C, Bennett MJ, Mooney SJ, Pridmore TP. **2013**. Recovering complete plant root system architectures from soil via X-ray μ-Computed Tomography. *Plant Methods* **9**: 8.

Miguel MA, Postma JA, Lynch JP. **2015**. Phene synergism between root hair length and basal root growth angle for phosphorus acquisition. *Plant Physiology* **167**: 1430–1439.

Monteith JL. **1964**. Evaporation and environment. *Symposia of the society for experimental biology* **19**: 205–234.

Mooney SJ, Pridmore TP, Helliwell J, Bennett MJ. **2012**. Developing X-ray computed tomography to non-invasively image 3-D root systems architecture in soil. *Plant and soil* **352**: 1–22.

Nielsen KL, Lynch JP, Jablokow AG, Curtis PS. **1994**. Carbon cost of root systems: an architectural approach. *Plant and Soil* **165**: 161–169.

Nielsen KL, Lynch JP, Weiss HN. **1997**. Fractal geometry of bean root systems: correlations between spatial and fractal dimension. *American Journal of Botany* **84**: 26–33.

Pagès L. **2000**. How to include organ interactions in models of the root system architecture? The concept of endogenous environment. *Annals of Forest Science* **57**: 535–541.

Pagès L, Vercambre G, Drouet JL, Lecompte F, Collet C, Le Bot J. **2004**. RootTyp: A generic model to depict and analyse the root system architecture. *Plant and Soil* **258**: 103–119.

Palmer CM, Bush SM, Maloof JN. **2001**. Phenotypic and developmental plasticity in plants. eLS. John Wiley & Sons, Ltd.

Penman HL. **1948**. Natural evaporation from open water, bare soil and grass. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences* **193**: 120–145.

Pierret A, Doussan C, Capowiez Y, Bastardie F, Pagès L. **2007**. Root functional architecture: A framework for modeling the interplay between roots and soil. *Vadose Zone Journal* **6**: 269–281.

Pigliucci M, Schlichting CD, Jones CS, Schwenk K. **1996**. Developmental reaction norms: the interactions among allometry, ontogeny and plasticity. *Plant Species Biology* **11**: 69–85.

Postma JA, Dathe A, Lynch JP. **2014a**. The optimal lateral root branching density for maize depends on nitrogen and phosphorus availability. *Plant Physiology* **166**: 590–602.

Postma JA, Jaramillo RE, Lynch JP. **2008**. Towards modeling the function of root traits for enhancing water acquisition by crops. In: Ahuja LR, Reddy VR, Saseendran SA, Yu Q, eds. Advances in Agricultural Systems Modeling. Response of Crops to Limited Water: Understanding and Modeling Water Stress Effects on Plant Growth Processes. Madison, Wis., USA: ASA-CSSA-SSSA, 251–276.

Postma JA, Lynch JP. **2011a**. Root cortical aerenchyma enhances the growth of maize on soils with suboptimal availability of nitrogen, phosphorus, and potassium. *Plant Physiology* **156**: 1190–1201.

Postma JA, Lynch JP. **2011b**. Theoretical evidence for the functional benefit of root cortical aerenchyma in soils with low phosphorus availability. *Annals of Botany* **107**: 829–841.

Postma JA, Lynch JP. **2012**. Complementarity in root architecture for nutrient uptake in ancient maize/bean and maize/bean/squash polycultures. *Annals of Botany* **110**: 521–534.

Postma JA, Schurr U, Fiorani F. **2014b**. Dynamic root growth and architecture responses to limiting nutrient availability: linking physiological models and experimentation. *Biotechnology Advances* **32**: 53–65.

Rubio G, Walk T, Ge Z, Yan X, Liao H, Lynch JP. **2001**. Root gravitropism and below-ground competition among neighbouring plants: A modelling approach. *Annals of Botany* **88**: 929–940.

Saengwilai P, Nord E, Chimungu J, Brown K, Lynch J. **2014**. Root cortical aerenchyma enhances nitrogen acquisition from low nitrogen soils in maize (Zea mays L.). *Plant Physiology* **166**: 726–735.

Sebastian J, Yee M-C, Viana WG, Rellán-Álvarez R, Feldman M, Priest HD, Trontin C, Lee T, Jiang H, Baxter I, *et al.* 2016. Grasses suppress shoot-borne roots to conserve water during drought. *Proceedings of the National Academy of Sciences* **113**: 8861–8866.

Šimunek J, Huang K, van Genuchten MT. **1995**. *The SWMS 3D code for simulating water flow and solute transport in three-dimensional variably-saturated media*. California: U. S. Salinity laboratory, USDA.

Stingaciu L, Schulz H, Pohlmeier A, Behnke S, Zilken H, Javaux M, Vereecken H. **2013**. In Situ Root System Architecture Extraction from Magnetic Resonance Imaging for Water Uptake Modeling. *Vadose Zone Journal* **12**: 9.

Vos J, Evers JB, Buck-Sorlin GH, Andrieu B, Chelle M, Visser PHB de. **2010**. Functional–structural plant modelling: A new versatile tool in crop science. *Journal of Experimental Botany* **61**: 2101–2115.

Walk TC, Jaramillo R, Lynch JP. **2006**. Architectural tradeoffs between adventitious and basal roots for phosphorus acquisition. *Plant and Soil* **279**: 347–366.

Walk TC, vanErp E, Lynch JP. **2004**. Modelling applicability of fractal analysis to efficiency of soil exploration by roots. *Annals of Botany* **94**: 119–128.

Wu L, McGechan MB, McRoberts N, Baddeley JA, Watson CA. **2007**. SPACSYS: Integration of a 3D root architecture component to carbon, nitrogen and water cycling--Model description. *Ecological Modelling* **200**: 343–359.

Wu Q, Pagès L, Wu J. **2016**. Relationships between root diameter, root length and root branching along lateral roots in adult, field-grown maize. *Annals of Botany* **117**:379–390.

Yang HS, Janssen BH. **2000**. A mono-component model of carbon mineralization with a dynamic rate constant. *European Journal of Soil Science* **51**: 517–529.

York LM, Galindo-Castañeda T, Schussler JR, Lynch JP. **2015**. Evolution of US maize (Zea mays L.) root architectural and anatomical phenes over the past 100 years corresponds to increased tolerance of nitrogen stress. *Journal of Experimental Botany* **66**: 2347–2358.

York LM, Silberbush M, Lynch JP. **2016**. Spatiotemporal variation of nitrate uptake kinetics within the maize (*Zea mays* L.) root system is associated with greater nitrate uptake and interactions with architectural phenes. *Journal of Experimental Botany* **67**: 3763–3775.

Zhan A, Schneider H, Lynch JP. **2015**. Reduced lateral root branching density improves drought tolerance in maize. *Plant Physiology* **168**: 1603–1615.

Zhang C, Postma JA, York LM, Lynch JP. **2014**. Root foraging elicits niche complementarity-dependent yield advantage in the ancient 'three sisters' (maize/bean/squash) polyculture. *Annals of Botany* **114**: 1719–1733.

Tables

Term	**Definition**
State variable	A quantity that has a unit and may depend on time and or space.
Minimodel	An object that encapsulates a state variable and is of a type derived from SimulaBase (Note S3). Minimodels place state variables in a context, give them a lifetime, a name, a unit and provide a general API for coupling of minimodels.
Module	A set of minimodels that form together a major component, like the carbon, nutrient or water modules.
Plugin	A class which adds functionality to the model without changing the main code (For example see Note S4). Plugins can be of derived type ObjectGenerator, DerivativeBase, or IntegrationBase
ObjectGenerator	plugin which instantiates new minimodels
DerivativeBase	Base classes for plugins that add new computational ability and/or new dependencies among minimodels
IntegrationBase	Base classes for plugins that add new integration procedures.
CLI	Command line interface, as opposed to a graphical user interface.
Root segment, root, root system, root system architecture (RSA)	Root segment is a short piece of root that can be represented by two coordinates, root is a single root axis, without branches, unless it stands in contrast to shoot, whereby it represents the whole root system (as in “root to shoot ratios”). Root system is a system of connected roots. Root system architecture is the spatio-temporal arrangement of the root system (Lynch, 1995) and is characterized by RSA traits such as branching frequencies or root gravitropism. RSA is often described by its geometric attributes, such as depth, width, specific root length, etc.

Table 1: Definition of terms.

Declaration of minimodel	**Explanation**
<SimulaDerivative name="root-GrowthRate" function="usePath" unit="cm/day">	Declaration of a minimodel named rootGrowthRate which uses the plugin “usePath” to simulate a growth rate with the unit cm/day.
<SimulaConstant name="path" type="string"> rootGrowth </Simula-Constant>	Declaration of a minimodel named “path” which contains a string of the path to which “rootGrowthRate” needs to be coupled
<SimulaConstant name="multiplier"> 0.1 </SimulaConstant>	Declaration of a minimodel named “multiplier” which is a simple constant with which the result of minimodel named by “path” should be multiplied with.
</SimulaDerivative>	Closing of the declaration of minimodel “rootGrowthRate”, so it is clear that “path” and “multiplier” are owned by it.
<SimulaVariable name="rootGrowth" function="useName+Rate" integra-tionFunction="RungeKutta4" unit="cm" > 1. </SimulaVariable>	Declaration of minimodel named rootGrowth, which will use function “useName+Rate” to retrieve data and will integrate that data with the default integration function, RungeKutta4. Start value is 1.

Table 2: A simple example of how a simple relative growth rate model can be constructed with *OpenSimRoot* by coupling two minimodels, one simulating the rate of growth (rgr=0.1*length), and one that integrates that rate (analytical result would be length=exp(0.1*t)). The rate calculation is done using the plugin “usePath” which simply retrieves the length using the declared path and uses the multiplier to calculate the fraction (0.1). The integration is done by the default integration method, RungeKutta4, which integrates the result computed by the plugin “useName+Rate”. This plugin simply retieves the values of minimodel “rootGrowthRate”. If the user would like the relative growth rate to be time dependent, the minimodel “multiplier” can be declared as an interpolation table, i.e. <SimulaTable name=”multiplier” …> 0 0.1 10 0.05 </SimulaTable>. Alternatively, stochasticity could be introduced by declaring the multiplier as of class SimulaStochastic. This model is obviously superfluous, and most plugins will implement more complex computations, with more dependencies (see also Note S6).

Figure legends

Figure 1: Schematic representation of the *OpenSimRoot* code. Code encompasses three major components, the command line interface (CLI), different types of minimodels and a library of plugins. The class hierarchy for each component is given in Note S3.

Figure 2: Simulated root system of bean (left) and maize (right) as rendered with ParaView. Root systems are made up of different root classes, each with their own root diameter, branching rules, growth direction and growth rates. Root cross-sections are not simulated but illustrate root segment traits that are represented in *OpenSimRoot*.

Figure 3: Schematic representation of the coupling of the Evapotranspiration, xylem transport and soil water modules. a) Soil pedon with the hydraulic head indicated in pseudo color (left) and three barley root systems (right) taking up water from that column. At the dry top water uptake is negative, meaning that some hydraulic lift occurred in this scenario. b) The Penman-Monteith equation for simulating transpiration and evaporation. c) Zoomed version of roots, showing the edges and vertices. d) Network model for simulating water flow through the roots. e) Water transport in three dimensions in the soil is simulated by solving the Richards equation, which combines Darcy’s law with mass conservation, using the Finite Element Method.

Figure 4: Schematic representation of the nutrient uptake, nutrient requirements and growth regulation modules. a) Root nutrient uptake coupled to model for solute transport in the soil. b) Schematic representation of the radial 1D Barber-Cushman model used for simulating P uptake. (c) summary of how the ratio between nutrient requirements and nutrient uptake determines plant physiology and/or growth.

Figure 5: Simulation results for plastic and non-plastic root systems. Root plasticity was defined as increasing lateral branching density with increasing nutrient availability. Phosphorus availability (left two root systems) was high in the top soil, causing branching density to be high in the top as well. At the same time, the reduced branching density deeper down, due to plasticity, allows the plant to grow the individual laterals longer. Pseudo colors show the local phosphorus availability. Nitrate moves throughout the soil, and thereby the plasticity effect is less pronounced and difficult to trace (right two root systems).

Figure 6: Simulation of imaged root phenotypes. a) Rendering of an MRI image of a two week old maize root system and b) the simulation of that root system by *OpenSimRoot* (right). Pseudo colors in 6b show the root segment age as estimated based on root topology, linear interpolation and the assumption that emergence of laterals takes two days. C) Rendering of segmented X-ray CT image

of a 10 day old wheat root system. Soil has been sliced to make roots visible. D) *OpenSimRoot* simulation of the predicted nitrate depletion zone of in C imaged root phenotype. We assumed an initially homogeneous distribution of Nitrate within the simulated soil domain. Pseudo colors show the nitrate concentration on a plane cut approximately through the center of the root system.

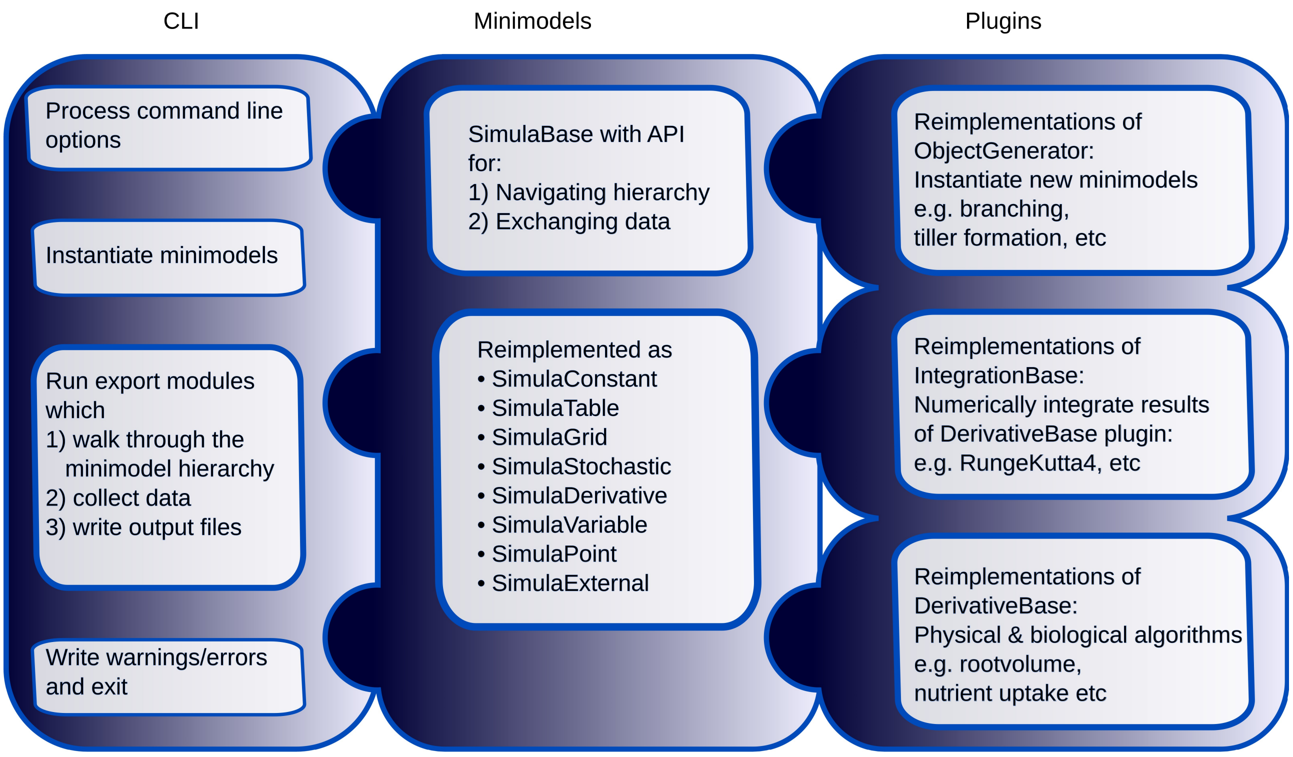


Figure 1: Schematic representation of the *OpenSimRoot* code. Code encompasses three major components, the command line interface (CLI), different types of minimodels and a library of plugins. The class hierarchy for each component is given in Note S3.

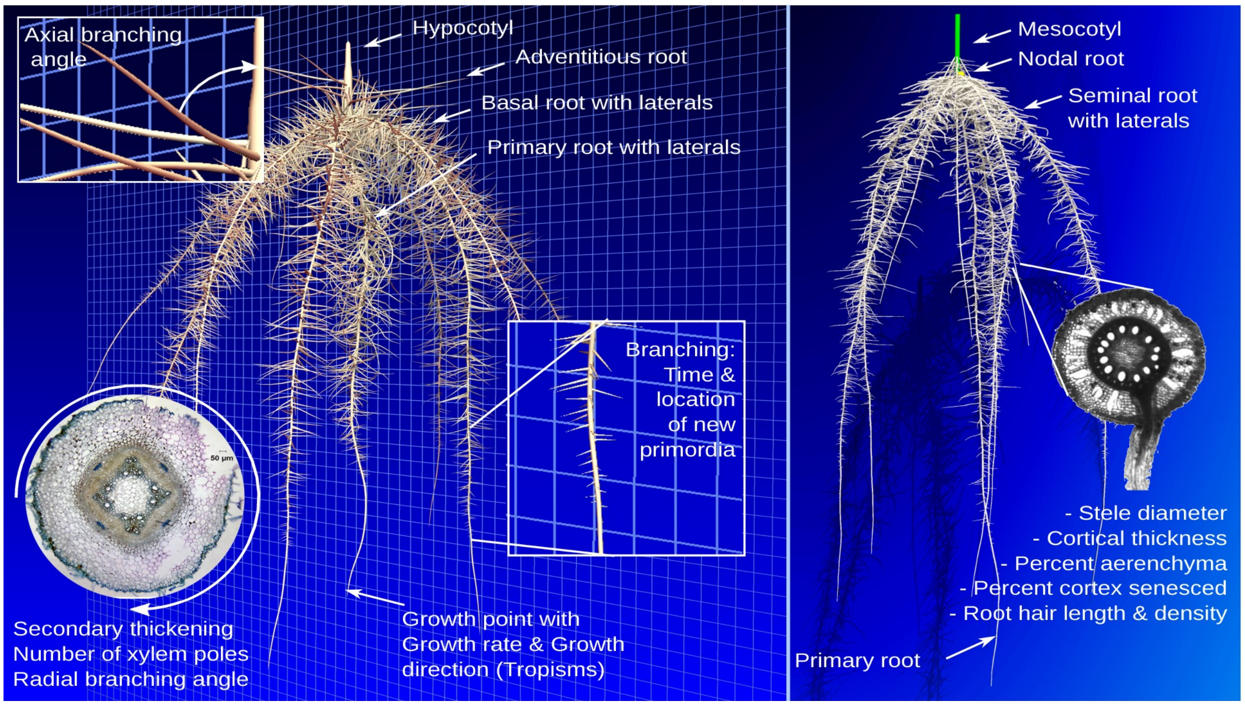


Figure 2: Simulated root system of bean (left) and maize (right) as rendered with ParaView. Root systems are made up of different root classes, each with their own root diameter, branching rules, growth direction and growth rates. Root cross-sections are not simulated but illustrate root segment traits that are represented in *OpenSimRoot*.

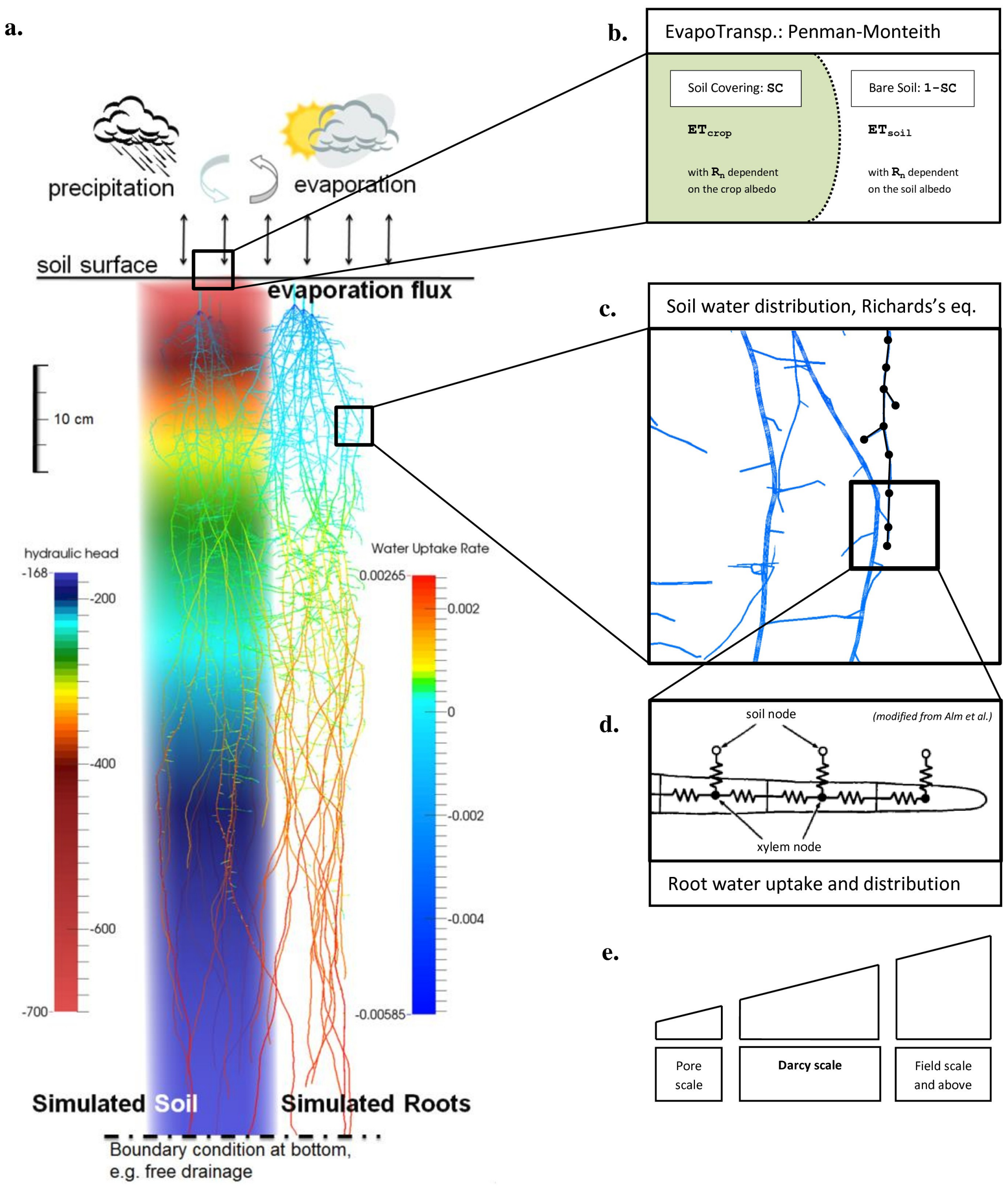


Figure 3: Schematic representation of the coupling of the Evapotranspiration, xylem transport and soil water modules. a) Soil pedon with the hydraulic head indicated in pseudo color (left) and three barley root systems (right) taking up water from that column. At the dry top water uptake is negative, meaning that some hydraulic lift occurred in this scenario. b) The Penman-Monteith equation for simulating transpiration and evaporation. c) Zoomed version of roots, showing the edges and vertices. d) Network model for simulating water flow through the roots. e) Water transport in three dimensions in the soil is simulated by solving the Richards equation, which combines Darcy's law with mass conservation, using the Finite Element Method.

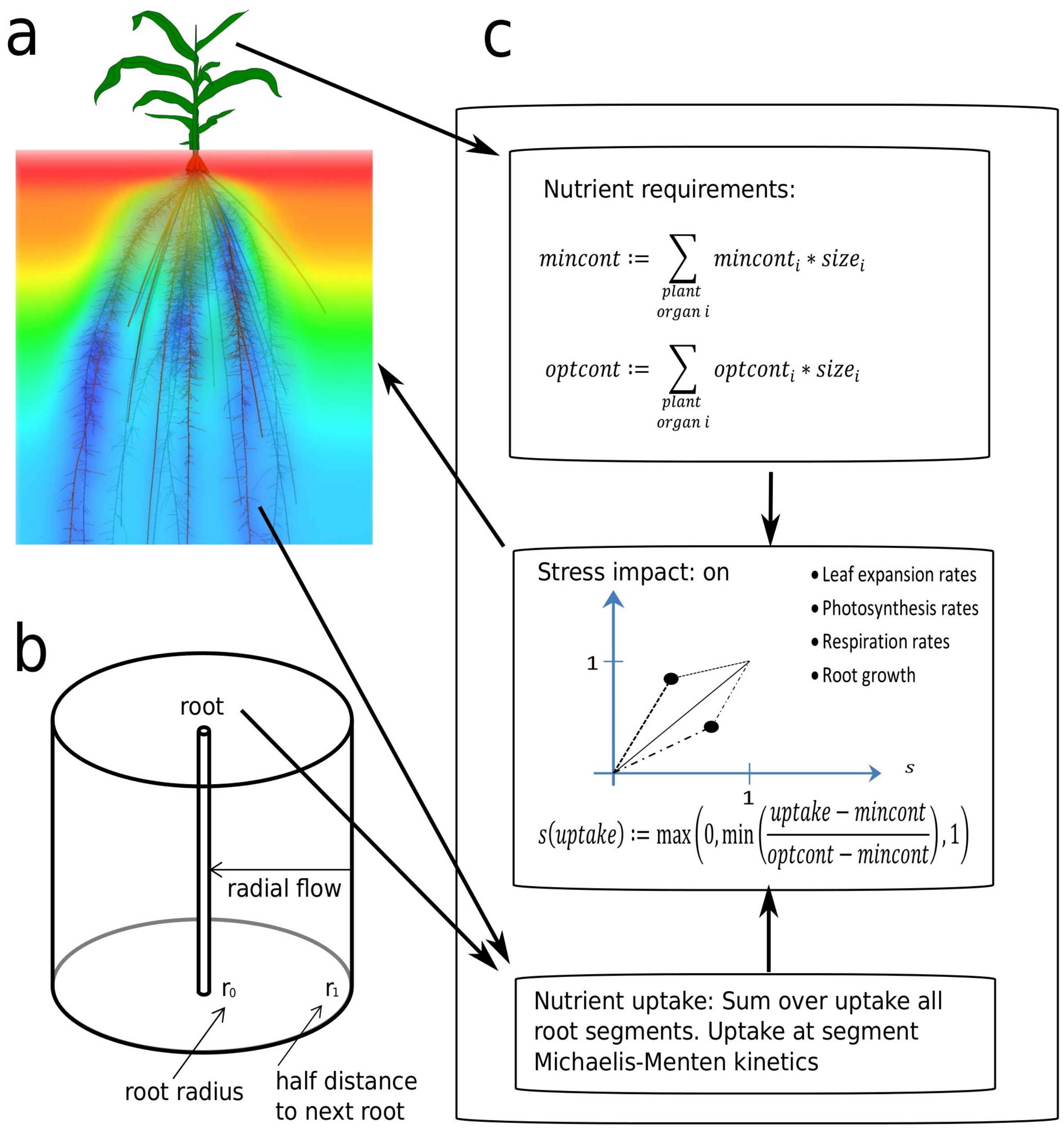

a
b
c
root
radial flow
r0
r1
root radius
half distance to next root
Nutrient requirements:
mincont := Σ_{plant organ i} mincont_i * size_i
optcont := Σ_{plant organ i} optcont_i * size_i
Stress impact: on
• Leaf expansion rates
• Photosynthesis rates
• Respiration rates
• Root growth
1
s
1
s(uptake) := max(0, min((uptake − mincont)/(optcont − mincont)), 1)
Nutrient uptake: Sum over uptake all root segments. Uptake at segment Michaelis-Menten kinetics

Figure 4: Schematic representation of the nutrient uptake, nutrient requirements and growth regulation modules. a) Root nutrient uptake coupled to model for solute transport in the soil. b) Schematic representation of the radial 1D Barber-Cushman model used for simulating P uptake. (c) summary of how the ratio between nutrient requirements and nutrient uptake determines plant physiology and/or growth.

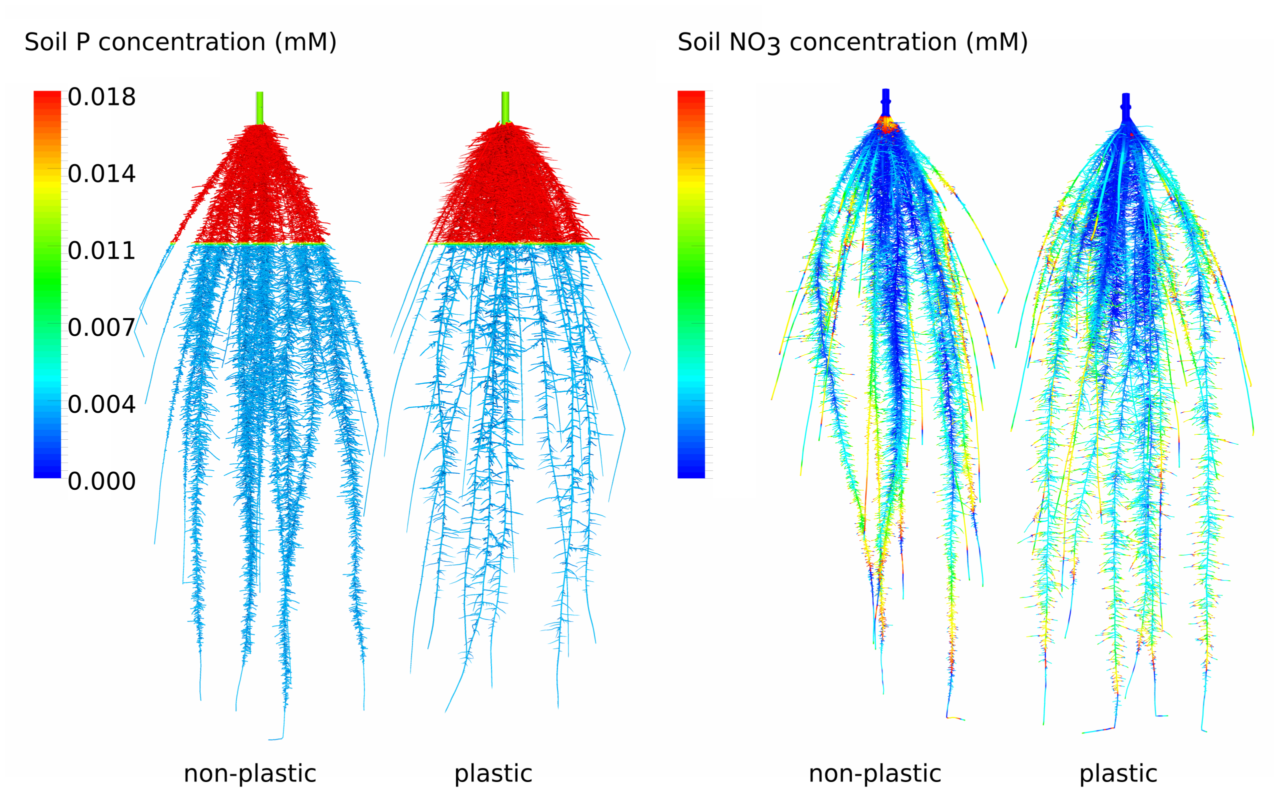


Figure 5: Simulation results for plastic and non-plastic root systems. Root plasticity was defined as increasing lateral branching density with increasing nutrient availability. Phosphorus availability (left two root systems) was high in the top soil, causing branching density to be high in the top as well. At the same time, the reduced branching density deeper down, due to plasticity, allows the plant to grow the individual laterals longer. Pseudo colors show the local phosphorus availability. Nitrate moves throughout the soil, and thereby the plasticity effect is less pronounced and difficult to trace (right two root systems).

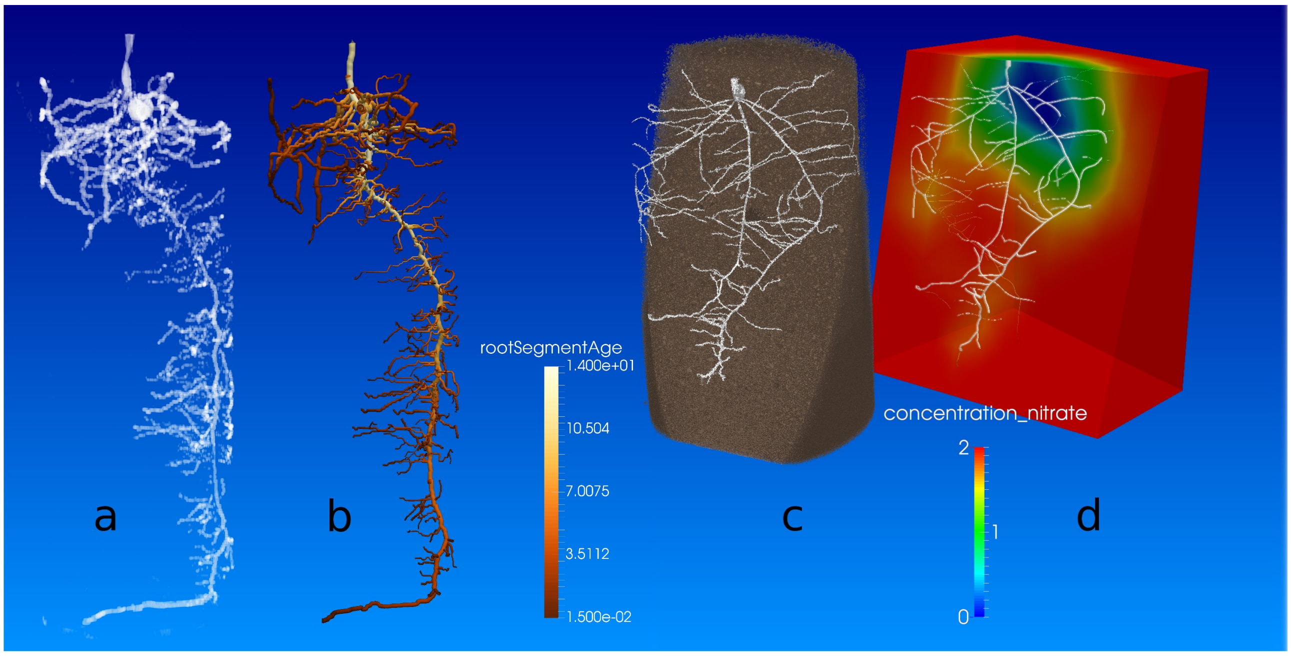


Figure 6: Simulation of imaged root phenotypes. a) Rendering of an MRI image of a two week old maize root system and b) the simulation of that root system by *OpenSimRoot* (right). Pseudo colors in 6b show the root segment age as estimated based on root topology, linear interpolation and the assumption that emergence of laterals takes two days. C) Rendering of segmented X-ray CT image of a 10 day old wheat root system. Soil has been sliced to make roots visible. D) *OpenSimRoot* simulation of the predicted nitrate depletion zone of in C imaged root phenotype. We assumed an initially homogeneous distribution of Nitrate within the simulated soil domain. Pseudo colors show the nitrate concentration on a plane cut approximately through the center of the root system.

New Phytologist Supporting Information

Article title: *OpenSimRoot*: Widening the scope and application of root architectural models

Authors: Postma, J.A.[1], Kuppe, C.[1] , Owen, M.R.[2,3], Mellor, N.[3,4], Griffiths, M.[3,4], Bennett, M.J.[3,4], Lynch J.P.[3,4,5], Watt, M. [1]

1) Plant Sciences, Institute of Bio and Geosciences 2, Forschungszentrum Jülich, Wilhelm-Johnen Straße 52425 Jülich, Germany
2) Centre for Mathematical Medicine and Biology, School of Mathematical Sciences, University of Nottingham, UK
3) Centre for Plant Integrative Biology, University of Nottingham, UK
4) Plant & Crop Sciences Division, School of Biosciences, University of Nottingham, UK
5) Department of Plant Science, Pennsylvania State University, USA

Article acceptance date:

The following Supporting Information is available for this article:

Supplement 1 Description of the SimulaBase API

Supplement 2 How to run *OpenSimRoot*: description of CLI

Supplement 3 Example C++ code for a plugin

Supplement 4 Example C++ code for a plugin

Supplement 5 Technical description of water and nutrient modules

Supplement 6 Example input file

Supplement 7 Example graph of state variables and their dependencies

Supplemental movie 1 Animation of Figure 6b.

Supplement 1: Application programming interface (API) of the SimulaBase class

This interface is used by the plugins to navigate the hierarchy and retrieve necessary data. For an example see, Supplement 4. Developers that would like to develop a new plugin, will need this interface in order to retrieve data from other minimodels. These minimodels are in a hierarchy. The methods listed here can be used to find those minimodels in the hierarchy, and to request data from them. Minimodels are instantiations (objects) of class (type) SimulaBase.

//Method to retrieve meta data on a given minimodel such as its name, path in the hierarchy, lifetime of the object, and its units.

```
std::string getName()const; //name of object
std::string getPrettyName()const; //some what more humen readable name
std::string getPath()const; //path to the object
virtual std::string getType()const; //What type this object has
bool evaluateTime(const Time &t)const; //check if t is within lifetime
Time getEndTime()const; //get the end time of object
Time getStartTime()const; //get the start time of object
virtual Unit getUnit(); //get the unit
void checkUnit(const Unit& unit)const; //check if unit equals given unit
void setUnit(const Unit &newUnit); //change unit
virtual void getXMLtag(Tag &tag); //get the object as tag (xml output)
```

//Methods to navigate the minimodel hierarchy

The difference between the get() and existing() methods is that when the object does not exist get() will throw an error and terminate the simulation, whereas existing() will return a NULL pointer. The getPath() methods will navigate a symbolic path just as a path in a filesystem is navigated. For example
getPath("../mySib") translates to getSibling("mysib"), where the later is more efficient.

```
SimulaBase* getParent()const;
SimulaBase* getParent(const unsigned int i) const;
int getNumberOfChildren()const;//does not update!
int getNumberOfChildren(const Time &t);//does update
SimulaBase* getChild(const std::string & name,const Time & t);
SimulaBase* existingChild(const std::string & name,const Time & t);
SimulaBase* getChild(const std::string & name);
SimulaBase* existingChild(const std::string & name);
SimulaBase* getChild(const std::string & name,const Unit & u);
SimulaBase* existingChild(const std::string & name,const Unit & u);
SimulaBase* getSibling(const std::string & name,const Time & t);
SimulaBase* existingSibling(const std::string & name,const Time & t);
SimulaBase* getSibling(const std::string & name);
SimulaBase* existingSibling(const std::string & name);
SimulaBase* getSibling(const std::string & name,const Unit & u);
SimulaBase* existingSibling(const std::string & name,const Unit & u);
//Sibling can be retrieved in alphabetic order.
SimulaBase* getNextSibling(const Time &t);
SimulaBase* getNextSibling()const;
```

```
    SimulaBase* getPreviousSibling(const Time &t);
    SimulaBase* getPreviousSibling()const;
    SimulaBase* getFirstChild(const Time &t);
    SimulaBase* getFirstChild()const;
    SimulaBase* getLastChild()const;

    SimulaBase* getPath(const std::string &name);
    SimulaBase* getPath(const std::string &name, const Time &t);
    SimulaBase* getPath(const std::string &name, const Unit &u);
    SimulaBase* existingPath(const std::string &name);
    SimulaBase* existingPath(const std::string &name, const Time &t);
    SimulaBase* existingPath(const std::string &name, const Unit &u);

    typedef std::vector<SimulaBase*> List;
    void getAllChildren(List&, const Time &t);
    void getAllChildren(List&)const;
```

//Method for walking along a root axis. Retrieves the minimodel with the same

```
    name associated with the next vertex.
    virtual SimulaBase* followChain(const Time & t);
```

//Methods to retrieve specific subsets of minimodels based on position

```
    typedef  std::multimap<Coordinate,SimulaBase*> Positions;
    static void getAllPositions(const Time & t, Positions& list);
    static void getAllPositions(Positions& list);
    void getYSlice(const Time &, const double, const double, Positions&);
    void getPositionsWithinRadius(const Time &, const Coordinate& c, const
    double & r, Positions&);
    void getPositionsInsideBox(const Time &, const Coordinate&, const
    Coordinate &, Positions&);
```

//Methods for retrieving data

```
    virtual void get(const Time &t, int &returnConstant);
    virtual void get(const Time &t, std::string &returnConstant);
    virtual void getRate(const Time &t, Time &var);
    virtual void get(const Time &t, Coordinate &point);
    virtual void get(const Time &t, MovingCoordinate &point);
    virtual void getAbsolute(const Time &t, Coordinate &point);
    virtual void getBase(const Time &t, Coordinate &point);
    virtual void getRate(const Time &t, Coordinate &point);
    virtual void getAbsolute(const Time &t, MovingCoordinate &point);
    virtual void get(int &returnConstant);
    virtual void get(std::string &returnConstant);
    virtual void get(bool &returnConstant);
    virtual void get(const Time &x, Time &y);
    virtual void get(Time &x);
    virtual void get(const Time &t, const Coordinate & pos, double &y);
    virtual void get(const Time &t, const Coordinate & pos, Coordinate &y);
    virtual void getRate(const Time &t, const Coordinate & pos, double &y);
    virtual void get(Coordinate &point);
```

```
virtual void getAverageRate(const Time &t1, const Time &t2, double &var);
virtual void getAverageRate(const Time &t1, const Time &t2, Coordinate
&var);

//reverse data look up: returns time that object was nearest to given
value or position. Only works if the object is not garbage collected
virtual void getTime(const Coordinate &p, Time &t, Time tmin=-1, Time
tmax=0);
virtual void getTime(const double &p, Time &t, Time tmin=-1, Time tmax=0);
```

//Method for setting data, probably only implemented for timetables.

```
virtual void set(const double &x, const double &y);
```

//Methods to retrieve info on timestepping of a minimodel

```
virtual Time &minTimeStep();
virtual Time &maxTimeStep();
virtual Time &preferedTimeStep();
virtual Time lastTimeStep();
```

//Methods to control garbage collection, which will basically clean up the simulation history

```
virtual void collectGarbage(const Time&); //clean up history
virtual void garbageCollectionOff(); //keep history of this object always
```

//Other methods

```
void stopUpdatefunction(); //When implementing an objectgenerator signal
it has
finished creating all objects for all times, and can be deleted.
static void updateAll(const Time &); //update whole tree
void updateRecursively(const Time &); //update subtree
static void signalMeAboutNewObjects(SimulaBase* me); //if plugin has the
addObject() implemented, it will be signaled when new objects are being
instantiated by any of the object generators.
```

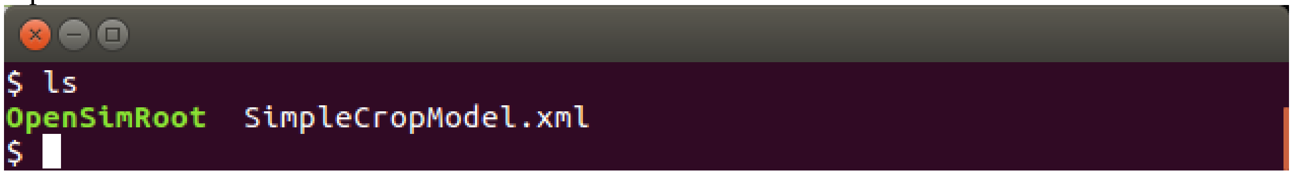
Supplement 2: Command line interface (CLI) of OpenSimRoot: How to run and use the model

OpenSimRoot has a command line interface, which means that you operate the model from a terminal using commands, not with a graphical interface and the mouse.

Step 1: Open a terminal (under windows 10 you may use the program named CMD)

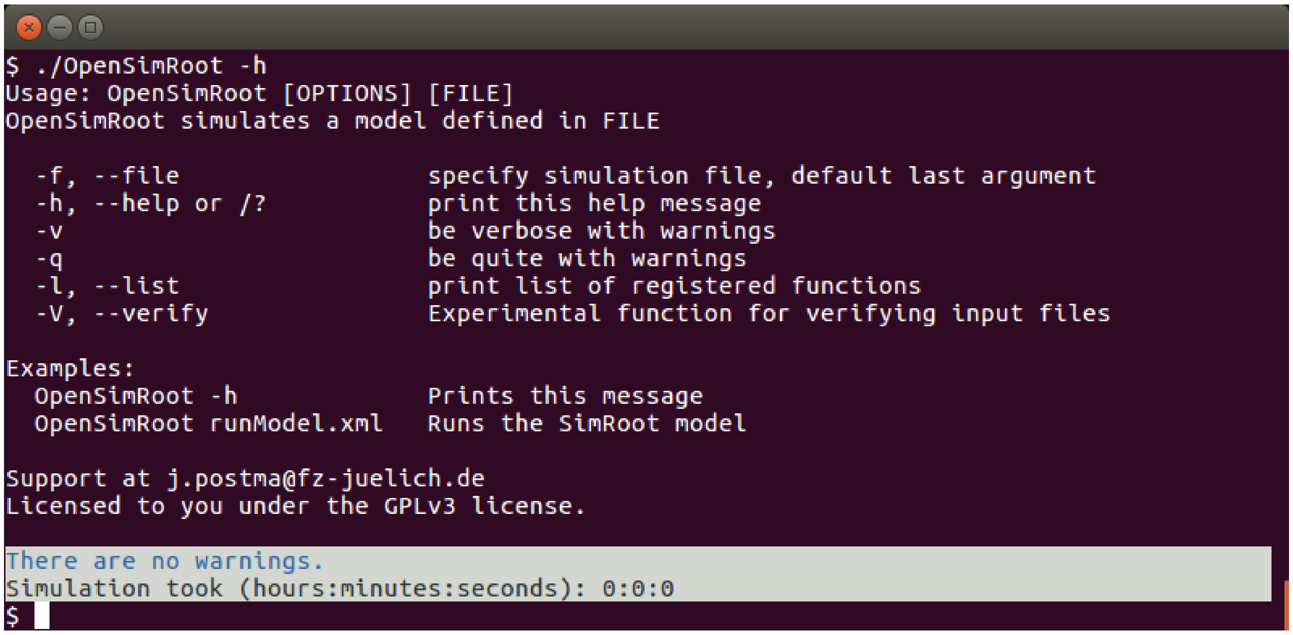
Step 2: Go to the folder where you want to run the model, use the command cd to navigate, for example: cd MyRunFolder

Step 3: We assume that the folder contains the OpenSimRoot executable. With the “ls” command you can list all folders (or on windows the command is “dir”). Here we see that my folder contains the executable OpenSimRoot (conveniently made green, as it is executable) and a XML input file.



```
$ ls
OpenSimRoot  SimpleCropModel.xml
$ 
```

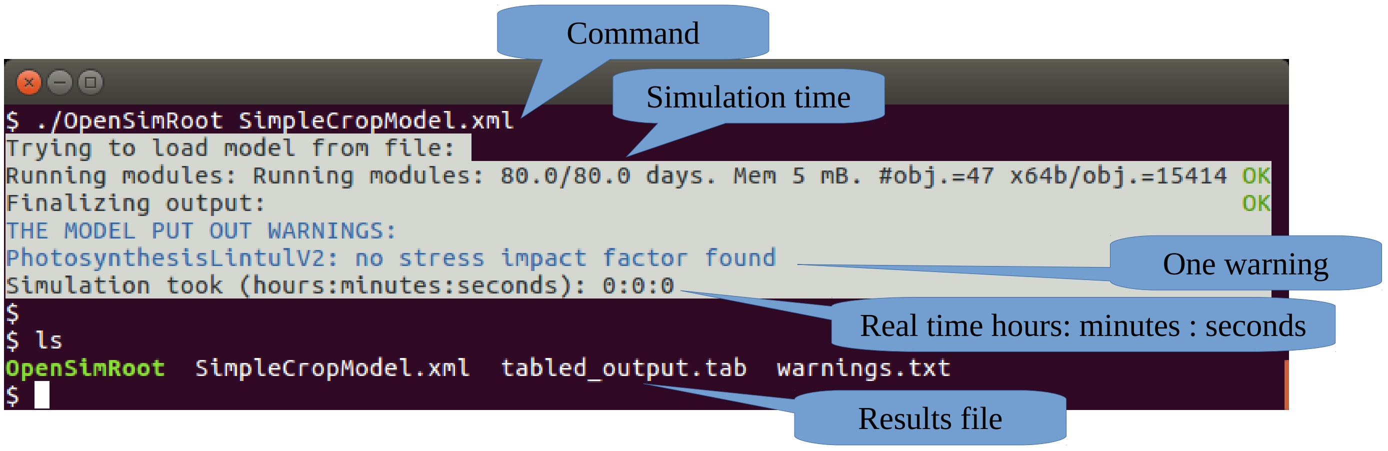
Step 4: OpenSimRoot has a small build in help which we we can run by typing ./OpenSimRoot -h (on windows you do not type the path “./” in front of the executable).



```
$ ./OpenSimRoot -h
Usage: OpenSimRoot [OPTIONS] [FILE]
OpenSimRoot simulates a model defined in FILE

  -f, --file                  specify simulation file, default last argument
  -h, --help or /?            print this help message
  -v                          be verbose with warnings
  -q                          be quite with warnings
  -l, --list                  print list of registered functions
  -V, --verify                Experimental function for verifying input files

Examples:
  OpenSimRoot -h              Prints this message
  OpenSimRoot runModel.xml    Runs the SimRoot model

Support at j.postma@fz-juelich.de
Licensed to you under the GPLv3 license.

There are no warnings.
Simulation took (hours:minutes:seconds): 0:0:0
$ 
```

The help shows how to run OpenSimRoot, and gives you some options and their explanation.

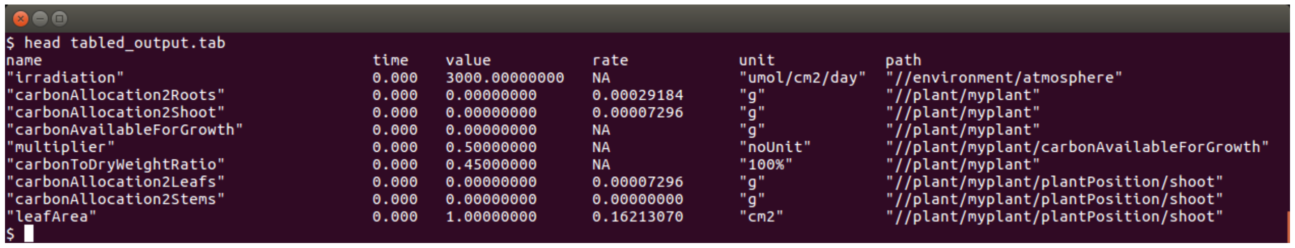
Step 5: Like the help shows, running the model is done by appending the input file: ./OpenSimRoot SimpleCropModel.xml



```
$ ./OpenSimRoot SimpleCropModel.xml
Trying to load model from file:
Running modules: Running modules: 80.0/80.0 days. Mem 5 mB. #obj.=47 x64b/obj.=15414 OK
Finalizing output:                                                                   OK
THE MODEL PUT OUT WARNINGS:
PhotosynthesisLintulV2: no stress impact factor found
Simulation took (hours:minutes:seconds): 0:0:0
$
$ ls
OpenSimRoot  SimpleCropModel.xml  tabled_output.tab  warnings.txt
$
```

Again with ls (dir) you can list the filer, the model created two new files, one containing warnings, one containing the simulation results.

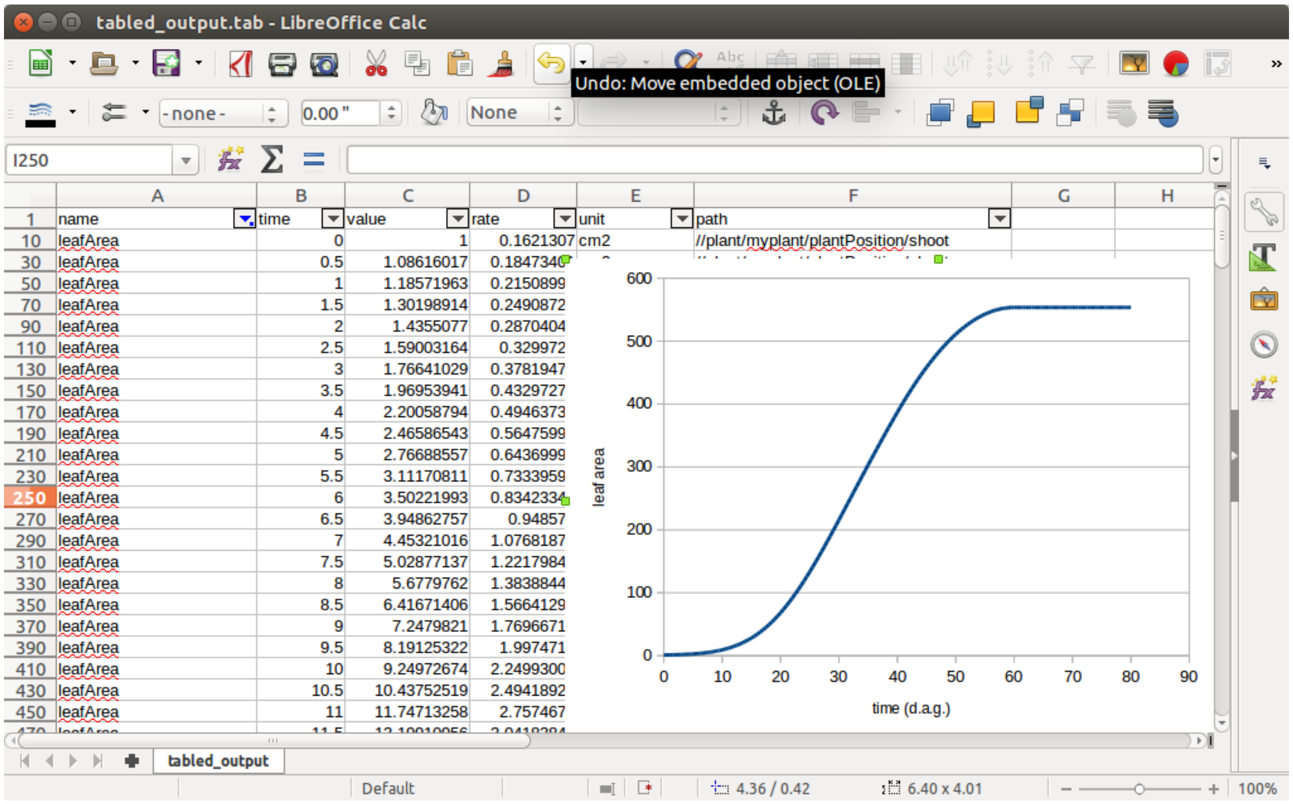
Step 6: The results of the simulation are in the tabled_output.tab file which can be viewed with any program that opens text files. Here we simply show the first lines with the command head:



```
$ head tabled_output.tab
name                                    time    value           rate            unit            path
"irradiation"                           0.000   3000.00000000   NA              "umol/cm2/day"  "//environment/atmosphere"
"carbonAllocation2Roots"                0.000   0.00000000      0.00029184      "g"             "//plant/myplant"
"carbonAllocation2Shoot"                0.000   0.00000000      0.00007296      "g"             "//plant/myplant"
"carbonAvailableForGrowth"              0.000   0.00000000      NA              "g"             "//plant/myplant"
"multiplier"                            0.000   0.50000000      NA              "noUnit"        "//plant/myplant/carbonAvailableForGrowth"
"carbonToDryWeightRatio"                0.000   0.45000000      NA              "100%"          "//plant/myplant"
"carbonAllocation2Leafs"                0.000   0.00000000      0.00007296      "g"             "//plant/myplant/plantPosition/shoot"
"carbonAllocation2Stems"                0.000   0.00000000      0.00000000      "g"             "//plant/myplant/plantPosition/shoot"
"leafArea"                              0.000   1.00000000      0.16213070      "cm2"           "//plant/myplant/plantPosition/shoot"
$
```

The file contains a header in the first row, and 6 columns listing the name of the state variable, the time, the value, the rate of change of that state variable (if simulated), the unit of the state variable, and the path in the hierarchy to this state variable.

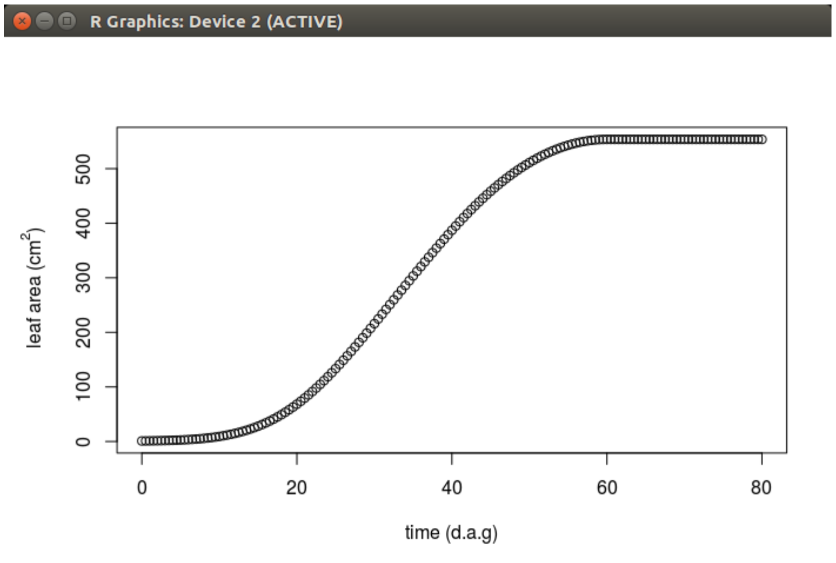
Step 7: The tabled_output.tab file is also easily imported into a spreadsheet program. By enabling the auto filter and selecting leafArea, we can easily create a plot.



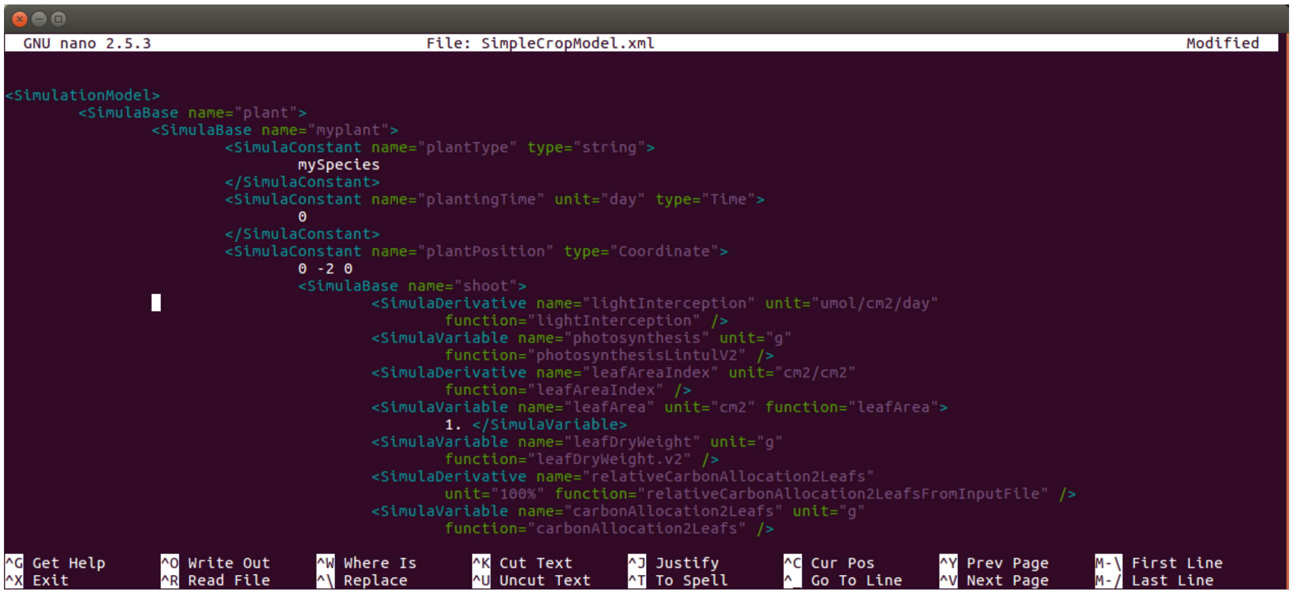
1	name	time	value	rate	unit	path
10	leafArea	0	1	0.1621307	cm2	//plant/myplant/plantPosition/shoot
30	leafArea	0.5	1.08616017	0.1847340		
50	leafArea	1	1.18571963	0.2150899		
70	leafArea	1.5	1.30198914	0.2490872		
90	leafArea	2	1.4355077	0.2870404		
110	leafArea	2.5	1.59003164	0.329972		
130	leafArea	3	1.76641029	0.3781947		
150	leafArea	3.5	1.96953941	0.4329727		
170	leafArea	4	2.20058794	0.4946373		
190	leafArea	4.5	2.46586543	0.5647599		
210	leafArea	5	2.76688557	0.6436999		
230	leafArea	5.5	3.11170811	0.7333959		
250	leafArea	6	3.50221993	0.8342334		
270	leafArea	6.5	3.94862757	0.94857		
290	leafArea	7	4.45321016	1.0768187		
310	leafArea	7.5	5.02877137	1.2217984		
330	leafArea	8	5.6779762	1.3838844		
350	leafArea	8.5	6.41671406	1.5664129		
370	leafArea	9	7.2479821	1.7696671		
390	leafArea	9.5	8.19125322	1.997471		
410	leafArea	10	9.24972674	2.2499300		
430	leafArea	10.5	10.43752519	2.4941892		
450	leafArea	11	11.74713258	2.757467		

8. The same can be achieved in R using this script:

```
        d<-read.table("tabled_output.tab",header=T)
        f=d$name=="leafArea"
        plot(value~time,data=d[f,],ylab=~"leaf area (cm"^2*")", xlab="time (d.a.g)")
```

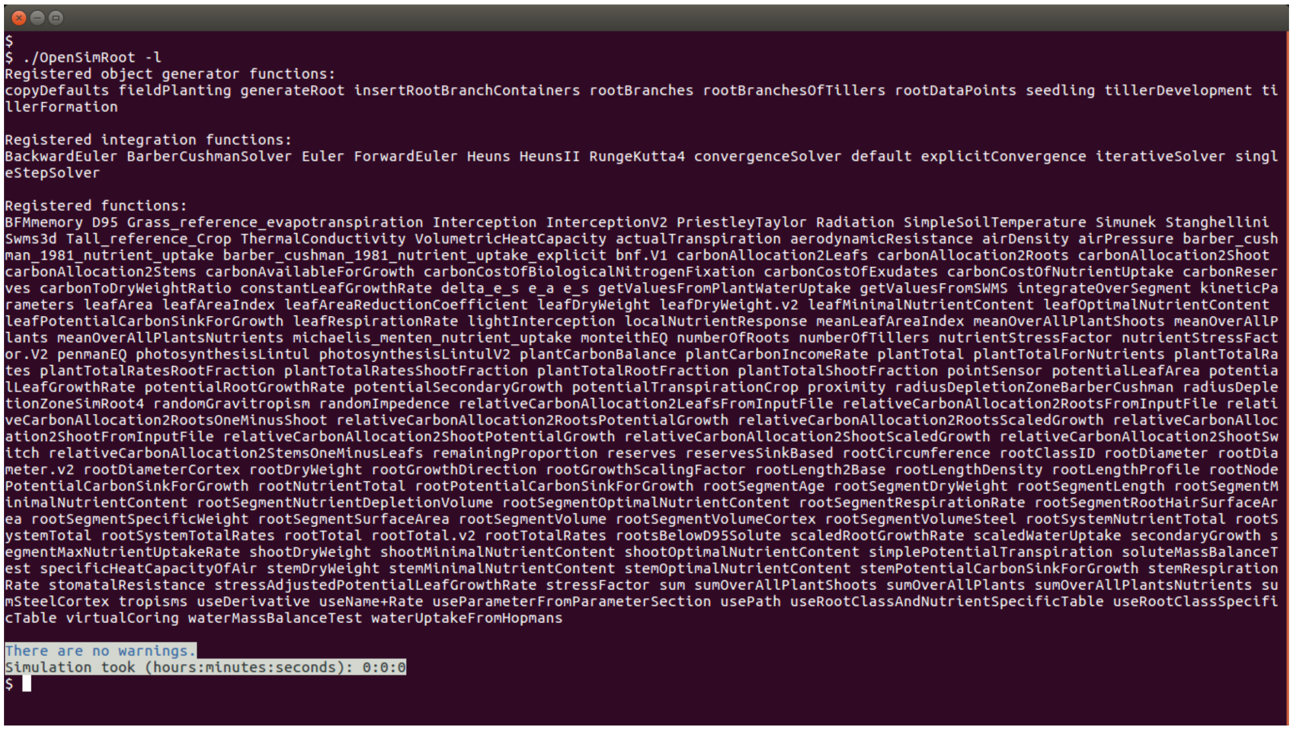


step 9: Editing the input file can be done with any text file editor. Here I opened the file with the command nano tabled_output.tab and the result is an xml formatted file in which we can change the numbers, save and rerun. In white you see the numbers, and scrolling to the bottom you would see more.



```
  GNU nano 2.5.3                        File: SimpleCropModel.xml                                     Modified


<SimulationModel>
        <SimulaBase name="plant">
                <SimulaBase name="myplant">
                        <SimulaConstant name="plantType" type="string">
                                mySpecies
                        </SimulaConstant>
                        <SimulaConstant name="plantingTime" unit="day" type="Time">
                                0
                        </SimulaConstant>
                        <SimulaConstant name="plantPosition" type="Coordinate">
                                0 -2 0
                                <SimulaBase name="shoot">
                                        <SimulaDerivative name="lightInterception" unit="umol/cm2/day"
                                                function="lightInterception" />
                                        <SimulaVariable name="photosynthesis" unit="g"
                                                function="photosynthesisLintulV2" />
                                        <SimulaDerivative name="leafAreaIndex" unit="cm2/cm2"
                                                function="leafAreaIndex" />
                                        <SimulaVariable name="leafArea" unit="cm2" function="leafArea">
                                                1. </SimulaVariable>
                                        <SimulaVariable name="leafDryWeight" unit="g"
                                                function="leafDryWeight.v2" />
                                        <SimulaDerivative name="relativeCarbonAllocation2Leafs"
                                                unit="100%" function="relativeCarbonAllocation2LeafsFromInputFile" />
                                        <SimulaVariable name="carbonAllocation2Leafs" unit="g"
                                                function="carbonAllocation2Leafs" />
^G Get Help      ^O Write Out     ^W Where Is      ^K Cut Text      ^J Justify       ^C Cur Pos       ^Y Prev Page     M-\ First Line
^X Exit          ^R Read File     ^\ Replace       ^U Uncut Text    ^T To Spell      ^_ Go To Line    ^V Next Page     M-/ Last Line
```

step 10: You see several functions listed that are used to simulate a state variable. To get a list of all functions that are included in your OpenSimRoot version use the command OpenSimRoot -l. This will list all plugins that are included with OpenSimRoot.



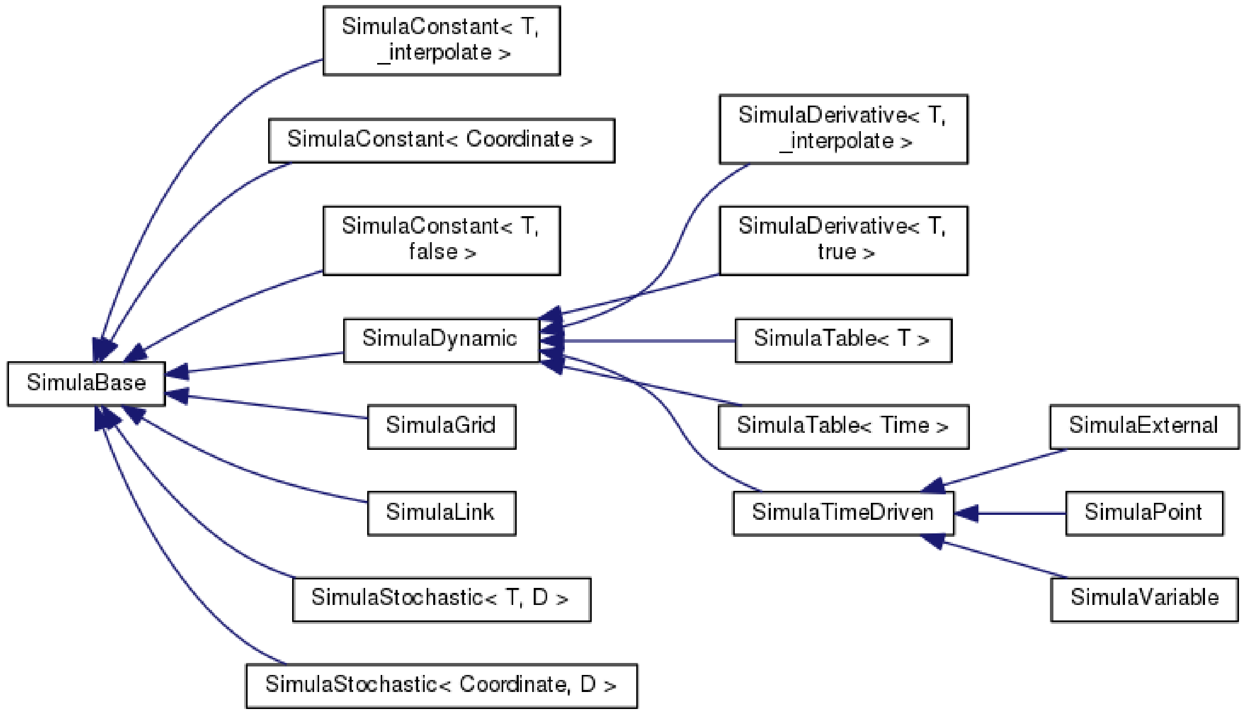
```
$
$ ./OpenSimRoot -l
Registered object generator functions:
copyDefaults fieldPlanting generateRoot insertRootBranchContainers rootBranches rootBranchesOfTillers rootDataPoints seedling tillerDevelopment ti
llerFormation

Registered integration functions:
BackwardEuler BarberCushmanSolver Euler ForwardEuler Heuns HeunsII RungeKutta4 convergenceSolver default explicitConvergence iterativeSolver singl
eStepSolver

Registered functions:
BFMmemory D95 Grass_reference_evapotranspiration Interception InterceptionV2 PriestleyTaylor Radiation SimpleSoilTemperature Simunek Stanghellini
Swms3d Tall_reference_Crop ThermalConductivity VolumetricHeatCapacity actualTranspiration aerodynamicResistance airDensity airPressure barber_cush
man_1981_nutrient_uptake barber_cushman_1981_nutrient_uptake_explicit bnf.V1 carbonAllocation2Leafs carbonAllocation2Roots carbonAllocation2Shoot
carbonAllocation2Stems carbonAvailableForGrowth carbonCostOfBiologicalNitrogenFixation carbonCostOfExudates carbonCostOfNutrientUptake carbonReser
ves carbonToDryWeightRatio constantLeafGrowthRate delta_e_s e_a e_s getValuesFromPlantWaterUptake getValuesFromSWMS integrateOverSegment kineticPa
rameters leafArea leafAreaIndex leafAreaReductionCoefficient leafDryWeight leafDryWeight.v2 leafMinimalNutrientContent leafOptimalNutrientContent
leafPotentialCarbonSinkForGrowth leafRespirationRate lightInterception localNutrientResponse meanLeafAreaIndex meanOverAllPlantShoots meanOverAllP
lants meanOverAllPlantsNutrients michaelis_menten_nutrient_uptake monteithEQ numberOfRoots numberOfTillers nutrientStressFactor nutrientStressFact
or.V2 penmanEQ photosynthesisLintul photosynthesisLintulV2 plantCarbonBalance plantCarbonIncomeRate plantTotal plantTotalForNutrients plantTotalRa
tes plantTotalRatesRootFraction plantTotalRatesShootFraction plantTotalRootFraction plantTotalShootFraction pointSensor potentialLeafArea potentia
lLeafGrowthRate potentialRootGrowthRate potentialSecondaryGrowth potentialTranspirationCrop proximity radiusDepletionZoneBarberCushman radiusDeple
tionZoneSimRoot4 randomGravitropism randomImpedence relativeCarbonAllocation2LeafsFromInputFile relativeCarbonAllocation2RootsFromInputFile relati
veCarbonAllocation2RootsOneMinusShoot relativeCarbonAllocation2RootsPotentialGrowth relativeCarbonAllocation2RootsScaledGrowth relativeCarbonAlloc
ation2ShootFromInputFile relativeCarbonAllocation2ShootPotentialGrowth relativeCarbonAllocation2ShootScaledGrowth relativeCarbonAllocation2ShootSw
itch relativeCarbonAllocation2StemsOneMinusLeafs remainingProportion reserves reservesSinkBased rootCircumference rootClassID rootDiameter rootDia
meter.v2 rootDiameterCortex rootDryWeight rootGrowthDirection rootGrowthScalingFactor rootLength2Base rootLengthDensity rootLengthProfile rootNode
PotentialCarbonSinkForGrowth rootNutrientTotal rootPotentialCarbonSinkForGrowth rootSegmentAge rootSegmentDryWeight rootSegmentLength rootSegmentM
inimalNutrientContent rootSegmentNutrientDepletionVolume rootSegmentOptimalNutrientContent rootSegmentRespirationRate rootSegmentRootHairSurfaceAr
ea rootSegmentSpecificWeight rootSegmentSurfaceArea rootSegmentVolume rootSegmentVolumeCortex rootSegmentVolumeSteel rootSystemNutrientTotal rootS
ystemTotal rootSystemTotalRates rootTotal rootTotal.v2 rootTotalRates rootsBelowD95Solute scaledRootGrowthRate scaledWaterUptake secondaryGrowth s
egmentMaxNutrientUptakeRate shootDryWeight shootMinimalNutrientContent shootOptimalNutrientContent simplePotentialTranspiration soluteMassBalanceT
est specificHeatCapacityOfAir stemDryWeight stemMinimalNutrientContent stemOptimalNutrientContent stemPotentialCarbonSinkForGrowth stemRespiration
Rate stomatalResistance stressAdjustedPotentialLeafGrowthRate stressFactor sum sumOverAllPlantShoots sumOverAllPlants sumOverAllPlantsNutrients su
mSteelCortex tropisms useDerivative useName+Rate useParameterFromParameterSection usePath useRootClassAndNutrientSpecificTable useRootClassSpecifi
cTable virtualCoring waterMassBalanceTest waterUptakeFromHopmans

There are no warnings.
Simulation took (hours:minutes:seconds): 0:0:0
$
```

Supplement 3: Class hierarchy of OpenSimRoot code

This document lists the class hierarchy for the most important classes in OpenSimRoot.

Minimodels

Minimodels are of type SimulaBase and encapsulate one time and location dependent state variable. The inhertance diagram for all SimulaX classes is given in Supplemental Figure 3.1.


SimulaConstant< T, _interpolate >
SimulaConstant< Coordinate >
SimulaConstant< T, false >
SimulaDynamic
SimulaBase
SimulaGrid
SimulaLink
SimulaStochastic< T, D >
SimulaStochastic< Coordinate, D >
SimulaDerivative< T, _interpolate >
SimulaDerivative< T, true >
SimulaTable< T >
SimulaTable< Time >
SimulaTimeDriven
SimulaExternal
SimulaPoint
SimulaVariable

Supplemental Figure 3.1: Inheritance diagram for all SimulaBase classes.

- SimulaConstant encapsulates a constant of various types.
- SimulaDerivative encapsulates an algorithm. Available algorithms are all the DerivativeBase derived plugins.
- SimulaTable encapsulate an array of time,value combinations. Values are interpolated.
- SimulaExternal provides a mechanism for encapsulating other dynamic simulation models.
- SimulaPoint simulates a point and its movement through space.
- SimulaVariable simulates a value and change over time using numerical integration.
- SimulaGrid simulates a static, 3D field using a list of Coordinates with values and a 3D interpolation algorithm
- SimulaLink simply bridges to another minimodel in the hierarchy of minimodels.
- SimulaStochastic draws numbers from a random number generator.

Inherited from DerivativeBase

Below is a list of all the plugins that directly, or indirectly, inherit from DerivativeBase and can be used by SimulaVariable, SimulaPoint or SimulaDerivative for computation.

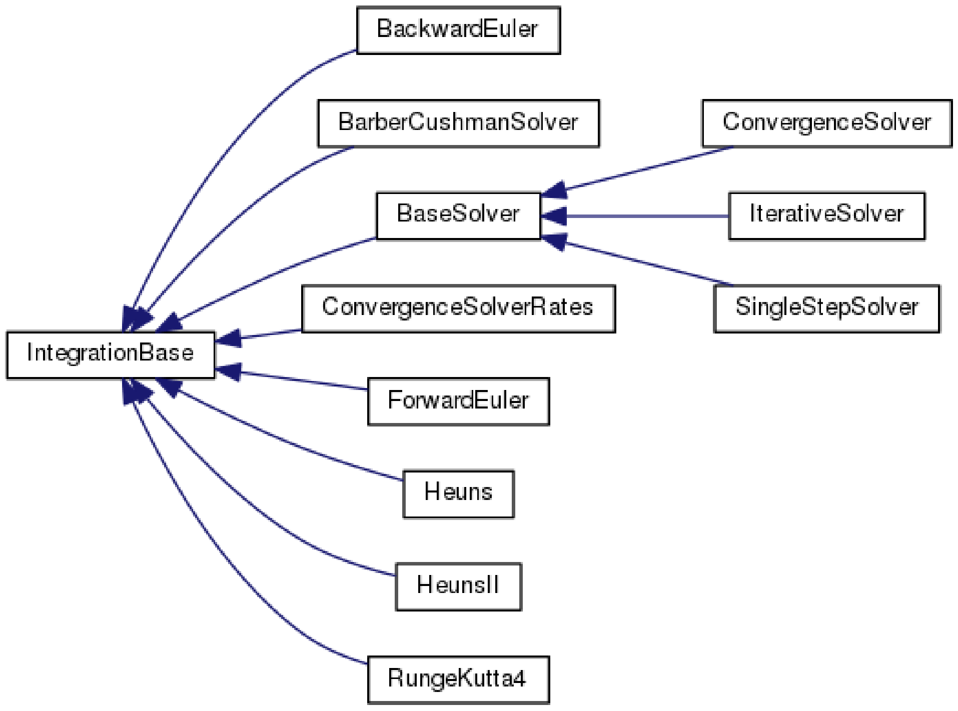
ActualTranspiration
ActualVaporPressure
AerodynamicResistance
AirDensity
AirPressure
BFMmemory
BiologicalNitrogenFixation
CarbonAllocation2Leafs
CarbonAllocation2Roots
CarbonAllocation2Shoot
CarbonAllocation2Stems
CarbonAvailableForGrowth
CarbonCostOfBiologicalNitrogenFixation
CarbonCostOfNutrientUptake
CarbonReserves
CinDryWeight
ConstantRootGrowthRate
D95
ETbaseclass
Grass_reference_evapotranspiration
Penman
PenmanMonteith
PriestleyTaylor
Stanghellini
Tall_reference_Crop
GetValuesFromPlantWaterUptake
GetValuesFromSWMS
Imax
Interception
InterceptionV2
LeafArea
LeafAreaIndex
LeafAreaReductionCoefficient
LeafDryWeight
LeafDryWeight2
LeafPotentialCarbonSinkForGrowth
LeafRespirationRate
LightInterception
LocalNutrientResponse
MeanLeafAreaIndex
NumberOfRoots
NumberOfTillers
NutrientStressFactor
NutrientStressFactorV2
PhotosynthesisLintul
PhotosynthesisLintulV2
PlantCarbonBalance
PlantCarbonIncomeRate
PlantTotal
PointSensor
PotentialLeafArea
PotentialTranspirationCrop
Proximity
Radiation
RadiusDepletionZoneBarberCushman
RadiusDepletionZoneSimRoot4
RandomGravitropism
RandomImpedence
RelativeCarbonAllocation2LeafsFromInputFile
RelativeCarbonAllocation2RootsFromInputFile
RelativeCarbonAllocation2RootsOneMinusShoot
RelativeCarbonAllocation2RootsPotentialGrowth
RelativeCarbonAllocation2RootsScaledGrowth
RelativeCarbonAllocation2ShootFromInputFile
RelativeCarbonAllocation2ShootPotentialGrowth
RelativeCarbonAllocation2ShootScaledGrowth
RelativeCarbonAllocation2ShootSwitch
RelativeCarbonAllocation2StemsOneMinusLeafs
RemainingProportion
Reserves
ReservesSinkBased
RootCircumference
RootClassID
RootDryWeight
RootGrowthDirection
RootGrowthScalingFactor
RootLength2Base
RootLengthDensity
RootLengthProfile
RootNodePotentialCarbonSinkForGrowth
RootPotentialCarbonSinkForGrowth
RootsBelowD95Solute
RootSegmentAge
RootSegmentRespirationRate
RootSegmentRootHairSurfaceArea
RootSegmentSpecificWeight
RootSystemTotal
RootTotal
RootTotal2
SaturatedVaporPressure
ScaledRootGrowthRate
ScaledWaterUptake
ShootDryWeight
ShootOptimalNutrientContent
SimplePotentialTranspiration
SimpleSoilTemperature
SlopeVaporPressure
SoluteMassBalanceTest
SpecificHeatCapacityOfAir
StemDryWeight
StemPotentialCarbonSinkForGrowth
StemRespirationRate
StomatalResistance
StressAdjustedPotentialLeafArea
StressFactor
SumCarbonCosts
SumOverPlants
SumOverPlantsShoot
SuperCoring
Swms3d
ThermalConductivity
TotalBase
CarbonCostOfExudates
CortexDiameter
IntegrateOverSegment
PotentialSecondaryGrowth
RootDiameter
RootSegmentDryWeight
RootSegmentLength
RootSegmentSurfaceArea
RootSegmentVolume
RootSegmentVolumeCortex
SecondaryGrowth
SumSteelCortex
TotalBaseLabeled
Barber_cushman_1981_nutrient_uptake
Barber_cushman_1981_nutrient_uptake_explicit
MichaelisMenten
OptimalNutrientContent
RootSegmentNutrientDepletionVolume
SegmentMaxNutrientUptakeRate
Tropisms
UseDerivative
UseParameterFromParameterSection
UseRootClassAndNutrientSpecificTable
VolumetricHeatCapacity
WaterMassBalanceTest
WaterUptakeFromHopmans

List of plugins for simulating various processes

Note that these are a list of classes, as they appear in the code. Registration of the plugins may occur under different names. Inputfiles use the registered names, not the class names. Use OpenSimRoot -l to get that list. See also operation manual in Supplement 2.

Integration functions

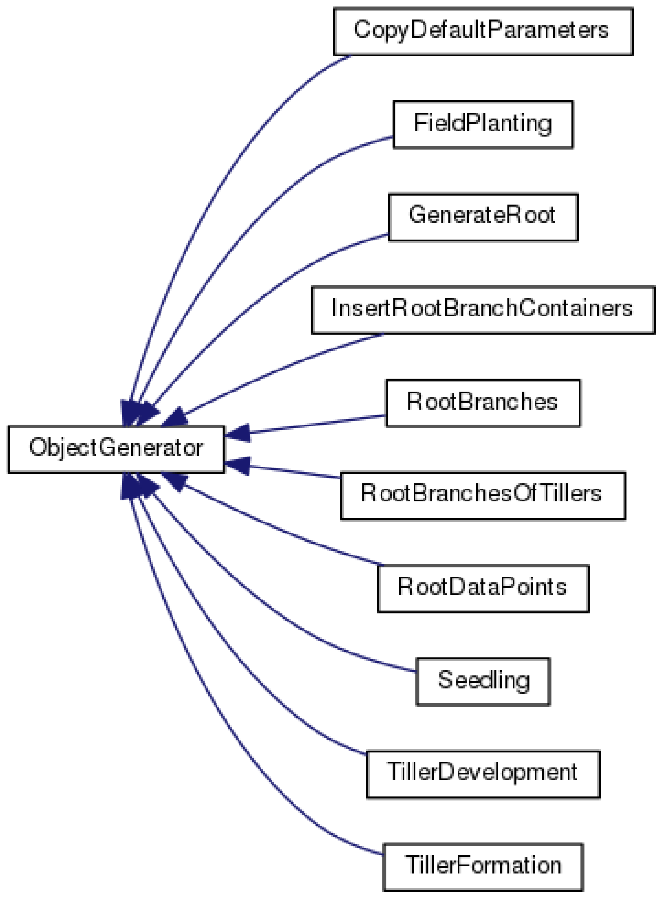
The SimulaVariable and SimulaPoint classes use helper functions for integrating the result. Several integration methods have been implemented (Supplemental figure 3.2). New integration functions can be added and registered, using the plugin framework, similar to the classes that inherit from DerivativeBase.



Supplemental figure 3.2: Inheritance diagram for the integration classes

Object generators

Object generators are plugins that can be associated with any SimulaX object and update the list of children when a child is requested.



Supplemental figure 3.3: Inheritance diagram for the object generators

Supplement 4: Plugin example code

Here we give example code for a simple plugin and the code needed to register this plugin with OpenSimRoot. Once the code has been put into a text file, it can be compiled and linked to OpenSimRoot.

1) For new algorithms

```
//Class declaration. Class should inherit from DerivativeBase, have a constructor, and
implements two virtual methods, getName() and calculate(). The example class presented here has
two SimulaBase pointers as private members, which will be used to connect to the minimodels that
simulate length and diameter of a root segment and to retrieve their values.

class RootSegmentSurfaceArea:public DerivativeBase{
public:
        RootSegmentSurfaceArea(SimulaDynamic* pSD);
        std::string getName()const;
protected:
        void calculate(const Time &t,double &var);
private:
        SimulaBase *diameter,*length;
};

//the constructor of our class. pSD is the pointer to the minimodel that uses the plugin for
computation
RootSegmentSurfaceArea::RootSegmentSurfaceArea(SimulaDynamic* pSD):DerivativeBase(pSD)
{
//We check that the user set the unit right
        pSD->checkUnit("cm2");
//We retrieve the pointers
        length=pSD->getSibling("rootSegmentLength","cm");
        diameter=pSD->getSibling("rootDiameter","cm");
}

//the computation
void RootSegmentSurfaceArea::calculate(const Time &t,double &area){
//first we retrieve data
        double d,l;
        diameter->get(t,d);
        length->get(t,l);
//second we compute
        area=l*d*PI;
}

//the  name under which the plugin will be registered, make sure it is unique, use OpenSimRoot
-l to see what names are already taken
std::string RootSegmentSurfaceArea::getName()const{
        return "rootSegmentSurfaceArea.v3";
}

//Now we create a function for instantiating our class
DerivativeBase * newInstantiationRootSegmentSurfaceArea(SimulaDynamic* const pSD){
   return new RootSegmentSurfaceArea(pSD);
}

//And we register this plugin using a static instantiation of a class which guarantees that the
constructor is when OpenSimRoot is started
static  class AutoRegisterMyNewPlugin {
public:
   AutoRegisterMyNewPlugin() {
//this line does the registration. Make sure you register under the same name as the getName()
method returns. This important for the model dump being loadable again.
        BaseClassesMap::getDerivativeBaseClasses()["rootSegmentSurfaceArea.v3"] =
newInstantiationRootSegmentSurfaceArea;
} rf9843hh923h; //the one static instance of this class
```

2) For new integration functions

```
//class declaration, must inherit from IntegrationBase, has a constructor,
// a getName() method and at least one integrate method
class BackwardEuler:public IntegrationBase{
public:
  BackwardEuler();
  std::string getName()const;
protected:
  virtual void integrate(SimulaVariable::Table & data, DerivativeBase & rateCalculator);
  virtual void integrate(SimulaPoint::Table & data, DerivativeBase & movementCalculator);
};

BackwardEuler::BackwardEuler():IntegrationBase(){}

void BackwardEuler::integrate(SimulaVariable::Table & data, DerivativeBase &rateCalculator){
  //...Your new algorithm here which should extend the data table, the derivative (rates) that
should be used are retrieved from the rateCalculator. For examples see code.
}

void BackwardEuler::integrate(SimulaPoint::Table & data, DerivativeBase & movementCalculator){
  //...Your new algorithm here, but then suitable for Coordinates, not doubles. Intended to
allow the simulation of a point moving through space. Mostly used to simulate the growth
trajectory of the root tip
};

std::string BackwardEuler::getName()const{
  return "BackwardEuler";
}

//function for instantiating the class
IntegrationBase * newInstantiationBackwardEuler(){
    return new BackwardEuler();
}

//Register the instantiation function
static class AutoRegisterIntegrationFunctions {
public:
  AutoRegisterIntegrationFunctions() {
    BaseClassesMap::getIntegrationClasses()["BackwardEuler"] = newInstantiationBackwardEuler;
  }
}p44608510843540385;//the one static instance of this class
```

3) For object generators

```
//class declaration for an object generator
class MyGenerator: public ObjectGenerator {
public:
  void initialize(const Time &t);
  void generate(const Time &t);
  MyGenerator(SimulaBase* const pSB);
};

//construction is delayed. Code is in the initialize method
MyGenerator::MyGenerator(SimulaBase* const pSB) :
  ObjectGenerator(pSB) {
}

//collecting of info, and or construction of minimodels at the start of the simulation
void MyGenerator::initialize(const Time &t) {
  //collect some info about planting time
  Time plantingTime;
  SimulaBase *pt=pSB->existingChild("plantingTime");
  if (pt) {
    //read planting time from file
    pt->get(t, plantingTime);
  } else {
    //copy from parent
    plantingTime = pSB->getStartTime();
  }

  //generate new plant by copying the template
  pSB->copyAttributes(plantingTime, ORIGIN->getChild("plantTemplate"));

  //we are done
  pSB->stopUpdatefunction();
}

void MyGenerator::generate(const Time &t) {
  //add code if there is time dependent generation of objects, not just at the start
}

//the function for instantiation of the class
ObjectGenerator * newInstantiationMyGenerator(SimulaBase* const pSB) {
  return new MyGenerator(pSB);
}

//register the instantiation function
static class AutoRegisterMyGeneratorInstantiationFunctions {
public:
  AutoRegisterMyGeneratorInstantiationFunctions() {
    BaseClassesMap::getObjectGeneratorClasses()["MyGenerator"] =
      newInstantiationMyGenerator;
  }
} p4595582386;
```

Supplement 5: Detailed description of the water and nutrient submodules

Watermodule

Plant transpiration is simulated by OpenSimRoot, assuming that water availability is not limiting and stomatal conductance is constant. Transpiration and evaporation need to be separated within OpenSimRoot. Transpiration can be estimated from a fixed water use efficiency parameter (which simply links carbon fixation linearly to transpiration), or from the Penman-Monteith model, which computes evapotranspiration based on weather conditions (Penman, 1948; Monteith, 1964). When transpiration is calculated based on a water use efficiency parameter, the user needs to provide evaporation values; when the Penman-Monteith model is used, transpiration and evaporation are separated by OpenSimRoot solving the Penman-Monteith model twice, once for full crop cover, and once for a bare soil. Based on the percent light capture by the crop OpenSimRoot scales evaporation and the transpiration terms assuming evaporation is negligible and small under full crop cover (Leaf Area Index ~3).

To simulate the soil hydrology, OpenSimRoot has a submodule that solves the Richards equation in three dimensions using finite element method (FEM) on a Cartesian grid. The soil water submodule is a simplified and modified C++ rewrite of the SWMS3D model, which is the basis of Hydrus and R-SWMS (Šimunek *et al.*, 1995; Diamantopoulos *et al.*, 2013).

Certain exceptional circumstances such as drainage or water ponding at top soil, are excluded. The top boundary condition is a water flux that is the difference between precipitation and evaporation. Evaporation, as computed by the Penman-Monteith equation, is assumed to be potential evaporation (i.e. appropriate for wet soils), and assumed to be equal across the soil surface, shoot geometry is not simulated. Potential evaporation is scaled back to an actual evaporation by including a smooth scaling function which causes evaporation to decrease smoothly from potential, when the top soil is wet, to equal the soil conductivity when the soil is not able to sustain higher evaporation rates. If the top soil is not necessarily uniformly wet, actual evaporation will be non-uniform across the soil surface in OpenSimRoot. The water retention curve and soil hydraulic conductivity are computed using the van Genuchten and Mualem equations.

The Richards equation can include a sink term, which in OpenSimRoot represents water uptake by roots (as described evaporation sink is handled as dynamic boundary condition). To do so we need to know 1) how much water is taken up by each root segment at a given moment in time, and 2) how that uptake is coupled to the FEM nodes of the grid on which the Richards equation is solved. Assuming that root uptake equals transpiration, i.e. we ignore temporal water storage in the plant, OpenSimRoot can either divide the water uptake of the whole root system by assuming each root segment contributes equally to uptake relative to its length (as in Hopmans, (Hopmans & Bristow, 2002)) or by solving the hyrdraulic architecture represented by a network model and using a circuit analogy likewise motivated by finite element theory (Alm *et al.*, 1992; Doussan *et al.*, 1998). The network model is novel in OpenSimRoot implemented to work with a growing root and used in the study of Schneider (Unpublished). This model requires axial and radial hydraulic conductivities for each root segment, which can be defined in the input files as a function of root age and class, and are scaled (i.e. normalized) with the inverse of the root segment length (axial), or the root segment surface area (radial). The coupling of the root model to the FEM model enables each root segment to have a soil water content at the root surface. The next step is to make sure that water uptake by the root system equals the transpiration which is

achieved by changing the water potential at the root collar (top of the hypocotyl). Getting the root collar potential is a parabolic optimization function which is solved with a newton solver, typically in three steps. The water potential at the top of the hypocotyl is not allowed to drop below a given threshold. If the threshold is reached, OpenSimRoot assumes that water uptake is less than potential transpiration and will write a warning. Further simulation results might not be correct as currently no effects of drought on photosynthesis, leaf expansion etc have been implemented.-However, the model should correctly deal with compensatory uptake of water when soil water distribution is heterogeneous. And this model can show water loss of roots while the same conductivity from xylem to soil is assumed.
Mapping the root model to the FEM model is done based on a neighborhood search. All FEM nodes surrounding the root segment are considered. Sink terms, and local environment are computed based on inverse distance weighted average of the FEM nodes surrounding the root node. An alternative mapping algorithm, by which every FEM node is assigned with every root node has been implemented, in order to ignore root architecture completely in the water and nutrient uptake simulations. This was for example used in Postma and Lynch (2012) where it was concluded that the positioning of the root, that is root architecture, is necessary for simulating niche differentiation for nitrate uptake among maize, bean and squash plants, whereas if roots would be able to take up nutrients from everywhere in the soil, there would be no niche differentiation.

Nutrient module
OpenSimRoot has a nutrient module to simulate the uptake solutes, and in the new version theoretically simultaneously for various nutrients. This module was implemented to simulate the function of root architectural traits for nutrient uptake, and test tradeoffs for acquisition of different nutrients. Time dependent optimal and minimal nutrient content (µmol/g) have to be defined for leaves, stems and all root classes, for to be simulated solutes. These amounts are used to compute nutrient requirements of the plant, and compared to total uptake amounts, including initial seed reserves (for uptake see below). When uptake is less than demand, plant stress is assumed, with maximum stress being defined as uptake equal to minimal nutrient content (stress(uptake) = max(0, min((uptake-minimal)/(optimal-minimal),1)). Stress modifying impact functions can be defined for components such as leaf expansion rate, photosynthesis rates, respiration rates, and root elongation rates or secondary growth. Typically, they should be defined such that, when stress=0, growth ceases altogether. For example, by making the initial response of the shoot stronger than that of the root, the plant will decrease shoot to root ratios when nutrient deficient. Thus OpenSimRoot will move towards a functional equilibrium, although due to the inherent slow nature of growth, and the relative fast dynamics of other processes, this functional equilibrium might not be reached, and oscillatory behavior might occur (Postma & Lynch, 2011; Postma *et al.*, 2014b). The current implementation assumes that internally, reallocation of nutrients is fast and perfect, such that all organs experience equal stress. This might be true for a nutrient like nitrogen, which typically causes chlorosis everywhere in the shoot, but might not be correct for other nutrients. The importance of simulation of nutrient redistribution in the plant still needs study, and would require implementation of a shoot architectural model in which the age and position of individual leafs is tracked.
Nutrient uptake from soil to root is simulated independently of utilization of nutrients within the plant. Two options for simulation are provided: 1) The Barber-Cushman model and 2) a 3D FEM

model. One is a C++ implementation of the original Barber-Cushman model with root hairs. The model is described as radial 1D PDE (Partial Differential Equation) which corresponds to the rhizosphere around the root. It assumes nutrient uptake to be described by a Michealis-Menten term, and the nutrient transport in the soil to be driven by convection (water flow) and diffusion. A buffer constant replaces a reaction term. The Barber-Cushman model is suitable for immobile nutrients like phosphorus. Phosphorus uptake causes steep gradients in concentrations around the root. These depletion zones are typically only 2-4 mm in diameter, and thereby would require a computationally unacceptably high resolution of the 3D finite element model (~0.1 mm resolution of a 1 m^3 soil pedon would result in $1e^{12}$ elements or 8 petabytes to hold a single double precision array).

Competition between roots is computed based on a local average root density which determines the outer boundary of the Barber-Cushman model. OpenSimRoot updates this boundary when new roots grow in the vicinity of other roots and corrects the initial nutrient concentration for new roots with the uptake of nutrients of older roots. Nevertheless, this handling of root competition is only acceptable when the overlap of depletion zones, which can be computed based on raster images of the root system, is relatively small. For crops, overlap in phosphorus depletion zones is typically below 20% because of its low mobility. Inter and intra root competition plays a much more important role in the uptake of mobile nutrients such as nitrate. Nitrate might form diffuse or no depletion zones around the root and for this reason is better simulated using a 3D FEM. SimRoot solved the convection-dispersion equation on the same FEM grid as the water transport is solved which can be restricting, OpenSimRoot alternatively can solve it on a refined grid, where the refinement factor is yet fixed to 2nd, 4th, 8th or the 16th of a reference grid. For each solute a new FEM model is instantiated and linked to the water model. The 3D FEM model for solute transport is coupled to the root systems using the same method as used for the hydraulic model, where the uptake of solutes by the root segments is based on Michaelis Menten kinetics, as in the Barber-Cushman model. Buffering and diffusion coefficients are dependent on the soil water content, and might thereby deviate from the constant coefficients used in the Barber-Cushman model. The effects must be considered when comparing the output of both models (Postma and Lynch, 2011).

When simulating more than one solute, solutes do not influence each other directly in OpenSimRoot. Indirect effects occur through the influence of nutrient uptake on root growth. Each solute has a stress function to determine how each impacts, for example, photosynthesis. A user specified aggregation function determines the aggregate impact (Dathe *et al.*, 2012). For example, Postma et al., (2014a) showed how the optimal lateral branching density in maize depends on the relative availability of phosphorus and nitrogen.

References

Alm DM, Cavelier J, Nobel PS. **1992**. A finite-element model of radial and axial conductivities for individual roots: development and validation for two desert succulents. *Annals of Botany* **69**: 87–92.

Dathe A, Postma JA, Lynch JP. **2012**. Modeling resource interactions under multiple edaphic stresses. In: Ahuja LR,, In: Reddy VR,, In: Saseendran SA,, In: Yu Q, eds. Advances in Agricultural Systems Modeling. Accepted for publication in. Madison, Wis., USA: ASA-CSSA-SSSA, .

Diamantopoulos E, Iden SC, Durner W. **2013**. Modeling non-equilibrium water flow in multistep outflow and multistep flux experiments. *HYDRUS Software Applications to Subsurface Flow and Contaminant Transport Problems*: 69.

Doussan C, Pagès L, Vercambre G. **1998**. Modelling of the hydraulic architecture of root systems: An integrated approach to water absorption - Model description. *Annals of Botany* **81**: 213–223.

Hopmans JW, Bristow KL. **2002**. Current capabilities and future needs of root water and nutrient uptake modeling. *Advances in Agronomy* **77**: 103–183.

Monteith JL. **1964**. Evaporation and environment. *Symposia of the society for experimental biology* **19**: 205–234.

Penman HL. **1948**. Natural evaporation from open water, bare soil and grass. Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences. The Royal Society, 120–145.

Postma JA, Dathe A, Lynch JP. **2014a**. The optimal lateral root branching density for maize depends on nitrogen and phosphorus availability. *Plant Physiology* **166**: 590–602.

Postma JA, Lynch JP. **2011**. Theoretical evidence for the functional benefit of root cortical aerenchyma in soils with low phosphorus availability. *Annals of Botany* **107**: 829–841.

Postma JA, Lynch JP. **2012**. Complementarity in root architecture for nutrient uptake in ancient maize/bean and maize/bean/squash polycultures. *Annals of Botany* **110**: 521–534.

Postma JA, Schurr U, Fiorani F. **2014b**. Dynamic root growth and architecture responses to limiting nutrient availability: linking physiological models and experimentation. *Biotechnology Advances* **32**: 53–65.

Šimunek J, Huang K, van Genuchten MT. **1995**. *The SWMS 3D code for simulating water flow and solute transport in three-dimensional variably-saturated media*. California: U. S. Salinity laboratory, USDA.

Supplement 6: Example of a simple OpenSimRoot input file

The XML below is an example of an OpenSimRoot input file that constructs a simple crop model, without any roots. All the SimulaX tags will instantiate a minimodel of the corresponding type, for example a constant (time independent parameter) is declared as <SimulaConstant ...>. Metadata for the minimodels, such as name and unit, are given in the attributes lists.

General rules for XML documents
1) The document has tags which are between brackets like <>
2) Tags correspond to minimodels in OpenSimRoot and therefore carry different names, such as SimulaBase, SimulaConstant, etc.
3) Tags need to be closed either by putting a / before the closing bracket, or if data is nested inside the tag with a corresponding closing tag which is recognized by </. For example <SimulaConstant></SimulaConstant>
4) Between opening and closing tags you will find data, and or declarations of minimodels which are at the next level in the hierarchy
5) Tags carry attributes which describe metadata. Attributes are always listed as attribute=”something”. In OpenSimRoot all tags have at least a name attribute.
6) An XML document is plain text and recognized by a special declaration at the top of the document. <?**xml** version="1.0" encoding="UTF-8"?>
7) XML documents can have stylesheets associated with them so the the browser knows how to render the document. Here we have <?xml-stylesheet type="text/xsl" href="tree-view.xsl"?>
8) Comments are between <!-- and -->.
9) All XML documents of a document type tag. For OpenSimRoot the document type is declared as <SimulationModel></SimulationModel>. All other tags must be in between these tags.

Here follows an example input file. The comments in black give more explanation as to how a simple crop model is being constructed by this input file. Input files for full root architectural models can be found in the software repository on gitlab:
https://gitlab.com/rootmodels/OpenSimRoot

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="tree-view.xsl"?>
<!--
Copyright © 2016 Forschungszentrum Jülich GmbH
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted
under the GNU General Public License v3 and provided that the following conditions are met:
1. Redistributions of source code must retain the above copyright notice, this list of conditions
and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of
conditions and the following disclaimer in the documentation and/or other materials provided with
the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to
endorse or promote products derived from this software without specific prior written permission.

Disclaimer
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

You should have received the GNU GENERAL PUBLIC LICENSE v3 with this file in license.txt but it
can also be found at http://www.gnu.org/licenses/gpl-3.0.en.html -->

<!--This XML constructs a simple, radiation use efficiency based crop model.

Roots and stems are only presented as Carbon (dry weight) pools
Leaf dry weight is converted to leaf area based on specific leaf area (SLA)
Leaf area is converted to light interception using an extinction coefficient.
Light interception is converted to photosynthesis using radiation use efficiency (RUE).
Photosynthesis is converted to structural carbon using a conversion factor (multiplier) which
represents relative losses due to respiration
Fixed allocation causes structural carbon to be divided over root, stem and leafs.

Behavior, is simple exponential growth for which
RGR = SLA * C2Leafs * photosynthesis * multiplier
However, as the light interception with increasing leaf area
reaches an asymptote, the model will move towards linear growth.-->

<SimulationModel>

<!-- SimulaBase is a simple container, that holds other SimulaX objects. SimulaBase is thus a
minimodel that does not hold or simulate data. It should, like all mini models, have a name. So
here we declare a container in which we are going to put all our plants. Inside it we put a
container for our plant, named arbitrarily “myPlant”.   -->

    <SimulaBase name="plants">
      <SimulaBase name="myplant">

<!-- Here follow three SimulaConstant declarations. SimulaConstant is a minimodel that holds time
and space independent data of different types. Possible types are double, int, string,
Coordinate. Besides the name attribute they must have a unit, and if the data is not a double, a
type declaration.

A plant should be of a given species/genotype. The model will look for a parameter set in
roottypeParameters with the corresponding type. Here we declare that we want to simulate a plant
of type mySpecies -->

        <SimulaConstant name="plantType" type="string">
            mySpecies
        </SimulaConstant>

<!-- The time that the plant is planted. 0. is at the start of the simulation. -->

        <SimulaConstant name="plantingTime" unit="day">
            0.
        </SimulaConstant>

<!-- Location in space where the seed is planted.  -->

        <SimulaConstant name="plantPosition" type="Coordinate">
            0 -2 0

<!-- Container that hold all the minimodels that will simulate shoot
related parameters. The shoot and root are inside plantPosition, as OpenSimRoot works with a
relative Coordinate system. We achieve that all coordinates that belong to our plant are relative
to plantPosition. -->

            <SimulaBase name="shoot">

<!-- Licht interception is simulated by the light interception module. SimulaDerivative declares
a minimodel that wil use the lightInterception plugin to compute light interception. Attributes
are name of what is being computed (name="lightInterception"), the unit of what is being computed
(unit="umol/cm2/day"), and the plugin that should be used to compute it
(function="lightInterception"). The plugin lightInterception requires leafAreaIndex and from the
parameter section and extinctionCoefficient (kdf). Further it needs irradiation levels from the
environmental section. All have been declared further down.  -->
```

```
              <SimulaDerivative name="lightInterception" unit="umol/cm2/day"
                function="lightInterception" />

<!--Simulation of photosynthesis rates can be done by the plugin registered as
photosynthesisLintulV2. However, since we want to know the total photosynthesis, the rates need
to be integrated over time. SimulaVariable does this. Thus unit is not g/day, but g. Attributes
are otherwise same as for a SimulaDerivative tag. Optional attributes that control the method of
integration and the timestep can be given. For example integrationFunction=”ForwardEuler” will
use the forward euler plugin for integrating. List of all integration methods can be obtained by
running OpenSimRoot -L. maximumTimeStep=”0.1” would reduce the maximum timestep from the default
0.2 to 0.1.
-->
              <SimulaVariable name="photosynthesis" unit="g"
                function="photosynthesisLintulV2" />

<!--Declaration of how leafAreaIndex should be simulated, as it is needed by the
lightInterception plugin. -->

              <SimulaDerivative name="leafAreaIndex" unit="cm2/cm2"
                function="leafAreaIndex" />

<!--Declaration of how leafArea should be simulated, as it is needed by the leafAreaIndex plugin.
Here the initial leaf area is given. More time value pairs can be entered in order to specify a
predefined initial leaf area. The leafArea plugin will simulate increases in leaf area on the
basis of carbon allocation to the leafs, the specificLeafArea and the carbonToDryweight ratio,
all declared later on.--->

              <SimulaVariable name="leafArea" unit="cm2" function="leafArea">
                0. 1. </SimulaVariable>

<!--Same as leafArea, but then for leafDryWeight. -->

              <SimulaVariable name="leafDryWeight" unit="g"
                function="leafDryWeight.v2"> 0. 0.001 </SimulaVariable>

<!--Here follow more minimodels, all with their respective plugins declared -->

              <SimulaDerivative name="relativeCarbonAllocation2Leafs"
                unit="100%"
                function="relativeCarbonAllocation2LeafsFromInputFile" />
              <SimulaVariable name="carbonAllocation2Leafs" unit="g"
                function="carbonAllocation2Leafs" />
              <!-- optional to have stem weight -->
              <SimulaDerivative name="relativeCarbonAllocation2Stems"
                unit="100%"
                function="relativeCarbonAllocation2StemsOneMinusLeafs" />
              <SimulaVariable name="carbonAllocation2Stems" unit="g"
               function="carbonAllocation2Stems" />
              <SimulaVariable name="stemDryWeight" unit="g"
               function="stemDryWeight" />
            </SimulaBase>
        </SimulaConstant>

<!--In this simulation it was decided to declare the carbonToDryWeight ratio as a simple
constant. -->

        <SimulaConstant name="carbonToDryWeightRatio" unit="100%">
            0.45
        </SimulaConstant>

<!--Carbon allocation -->

        <SimulaDerivative name="relativeCarbonAllocation2Shoot"
            unit="100%"
            function="relativeCarbonAllocation2ShootFromInputFile" />
        <SimulaVariable name="carbonAllocation2Shoot" unit="g"
            function="carbonAllocation2Shoot" />
```

```
<!--Instead of using a process specific plugin to simulate the carbon available for growth, here
we use a general plugin named usePath which simply couples the carbon available for growth to
photosynthesis. Since this declaration as a child called “multiplier” the photosynthesis rates is
halved, so it is assumed that half of all carbon fixed by photosynthesis is converted to plant
dry mass, the rest is respired. -->

        <SimulaDerivative name="carbonAvailableForGrowth"
            unit="g" function="usePath">
            <SimulaConstant name="path" type="string">
              plantPosition/shoot/photosynthesis
            </SimulaConstant>
            <!-- half of carbon assumed to be respired -->
            <SimulaConstant name="multiplier">0.5</SimulaConstant>
        </SimulaDerivative>

<!--Some declarations related to roots -->

      <SimulaDerivative name="relativeCarbonAllocation2Roots"  unit="100%"
               function="relativeCarbonAllocation2RootsOneMinusShoot" />
      <SimulaVariable name="carbonAllocation2Roots" unit="g"
               function="carbonAllocation2Roots" />
      <SimulaVariable name="rootDryWeight" unit="g" function="rootDryWeight" />

<!--The closing tags for the myPlant and Plants containers. -->

      </SimulaBase>
    </SimulaBase>



<!-- Environmental data needs to be declared, here all we need is irradiation in order to know
how much light is being captured for photosynthesis -->

    <SimulaBase name="environment">
      <SimulaBase name="atmosphere">
        <SimulaTable name_column1="time" name_column2="irradiation"
            unit_column1="day" unit_column2="umol/cm2/day">
            0 3000
            100 3000
        </SimulaTable>
      </SimulaBase>
    </SimulaBase>

<!-- here a parameter section for our plant is specified. -->

    <SimulaBase name="rootTypeParameters">
      <SimulaBase name="mySpecies">
        <SimulaBase name="resources">

<!--relativeCarbonAllocation to leafs (see above) uses a plugin in that simply looks up data from
a table. The table is declared here. SimulaTables have two columns. Each column has a name and a
unit declared in the attribute list. Here, as will be often the case, the first column is time.
This is time since the plant started growing, not since the start of the simulation. First all
carbon that is going to the shoot is allocated to leafs, later on more carbon is going to the
stems. Values in the table are interpolated linearly, unless a different interpolation method is
declared. Currently, only interpolation=”step” is implemented as alternative method.  -->

            <SimulaTable name_colum1="time" unit_colum1="day"
              name_colum2="carbonAllocation2LeafsFactor" unit_colum2="100%">
              0 1
              10 0.8
              40 0.5
              60 0.
              80 0.
            </SimulaTable>

<!--How much carbon should go to the root. The rest goes to the shoot. -->

            <SimulaTable name_colum1="time" unit_colum1="day"
              name_colum2="carbonAllocation2RootsFactor" unit_colum2="100%">
              0 0.8
              10 0.2
              40 0.2
              80 0.2
            </SimulaTable>
        </SimulaBase>

<!--Declaration of several well known shoot related parameters. -->

        <SimulaBase name="shoot">
            <SimulaConstant name="areaPerPlant" unit="cm2">
              100
            </SimulaConstant>
            <SimulaConstant name="extinctionCoefficient" unit="noUnit">
              0.6
            </SimulaConstant>
            <SimulaConstant name="lightUseEfficiency" unit="g/umol">
              0.4E-6
            </SimulaConstant>
            <SimulaTable name_colum1="time" name_colum2="specificLeafArea"
              unit_colum1="day" unit_colum2="g/cm2" note="SLA in lintul">
              0 0.001
              10 0.002
              40 0.003
              80 0.003
            </SimulaTable>
        </SimulaBase>
      </SimulaBase>
    </SimulaBase>

<!--This section gives the user some control over the output.-->

    <SimulaBase name="simulationControls">
      <SimulaBase name="outputParameters">
        <SimulaBase name="table">

<!--A table should be written containing values for each minimodel, for every half day from day 0
to 80. Hierarchy will be traversed up to depth 10 -->

            <SimulaConstant name="run" type="bool"> 1 </SimulaConstant>
            <SimulaConstant name="searchingDepth" type="int"> 10
            </SimulaConstant>
            <SimulaConstant name="startTime" type="time"> 0.
            </SimulaConstant>
            <SimulaConstant name="endTime" type="time"> 80.
            </SimulaConstant>
            <SimulaConstant name="timeInterval" type="time"> 0.5
            </SimulaConstant>
        </SimulaBase>
      </SimulaBase>
    </SimulaBase>

<!--We are done -->

</SimulationModel>
```

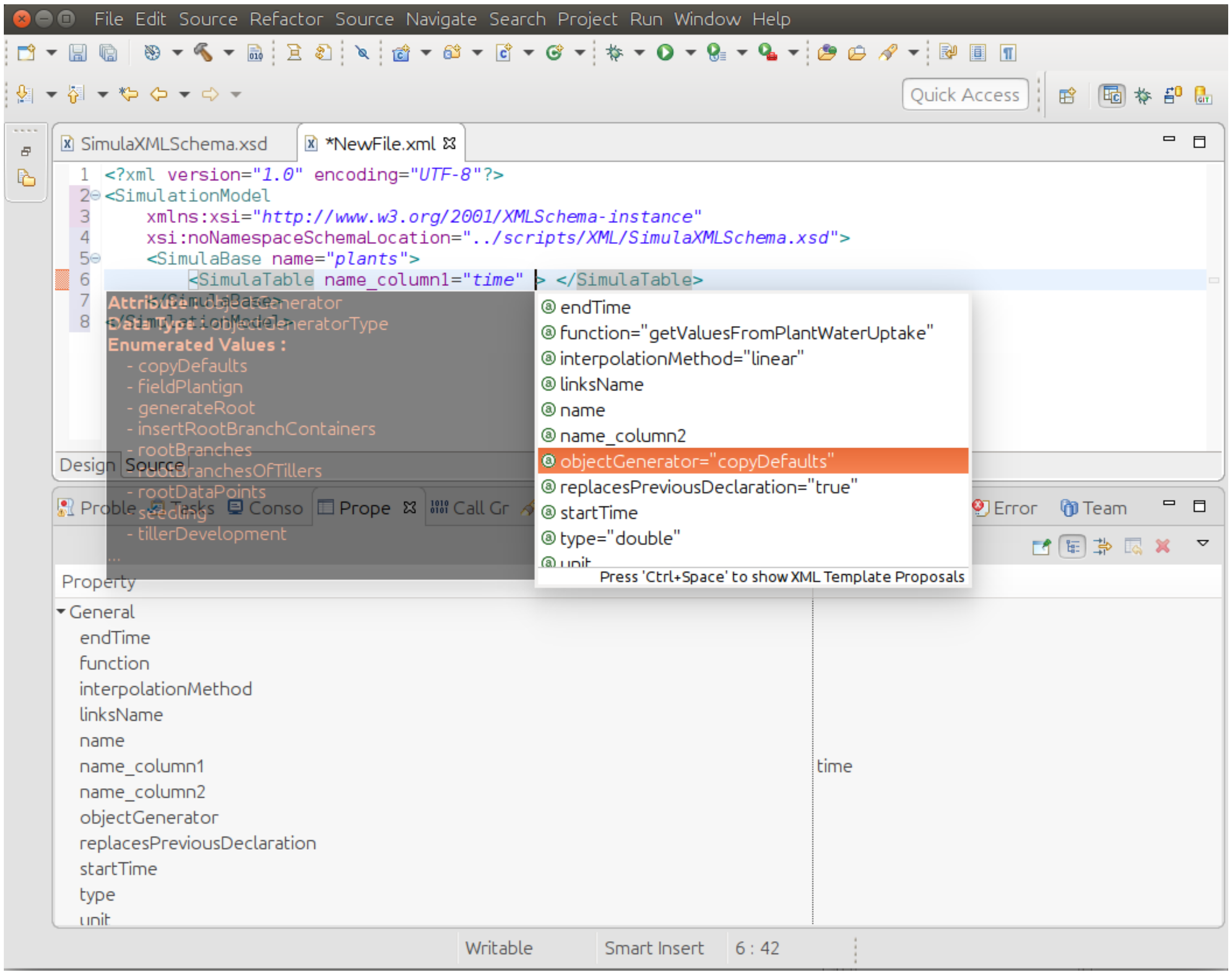
User friendly viewing of XML input files

A webbrowser can transform this into more human friendly presentation using the attached tree-view.xsl transformation style sheet (available for download at the gitlab repository https://gitlab.com/rootmodels/OpenSimRoot). The result when you open this file in a browser is given below.

```
|___ Origin
     |___  'plant'
     |     |___  'myplant'
     |           |___  'plantType' = mySpecies
     |           |___  'plantingTime' = 0 (day)
     |           |___  'plantPosition' = 0 -2 0
     |           |     |___  'shoot'
     |           |           |___  'lightInterception' (umol/cm2/day)
     |           |           |___  'photosynthesis' (g)
     |           |           |___  'leafAreaIndex' (cm2/cm2)
     |           |           |___  'leafArea' (cm2) initial value = 1.
     |           |           |___  'leafDryWeight' (g)
     |           |           |___  'relativeCarbonAllocation2Leafs' (100%)
     |           |           |___  'carbonAllocation2Leafs' (g)
     |           |           |___  'relativeCarbonAllocation2Stems' (100%)
     |           |           |___  'carbonAllocation2Stems' (g)
     |           |           |___  'stemDryWeight' (g)
     |           |___  'carbonToDryWeightRatio' = 0.45 (100%)
     |           |___  'relativeCarbonAllocation2Shoot' (100%)
     |           |___  'carbonAllocation2Shoot' (g)
     |           |___  'carbonAvailableForGrowth' (g)
     |           |     |___  'path' = plantPosition/shoot/photosynthesis
     |           |     |___  'multiplier' = 0.5
     |           |___  'relativeCarbonAllocation2Roots' (100%)
     |           |___  'carbonAllocation2Roots' (g)
     |           |___  'rootDryWeight' (g)
     |___  'environment'
     |     |___  'atmosphere'
     |           |___  x,y pairs :{ 0 3000 100 3000 }
     |___  'rootTypeParameters'
     |     |___  'mySpecies'
     |           |___  'resources'
     |           |     |___  'carbonAllocation2LeafsFactor' (100%)=f{'time'} (day) x,y pairs :{ 0 1 10 0.8 40 0.5 60 0. 80 0. }
     |           |     |___  'carbonAllocation2RootsFactor' (100%)=f{'time'} (day) x,y pairs :{ 0 0.8 10 0.2 40 0.2 80 0.2 }
     |           |___  'shoot'
     |                 |___  'areaPerPlant' = 100 (cm2)
     |                 |___  'extinctionCoefficient' = 0.6 (noUnit)
     |                 |___  'lightUseEfficiency' = 0.4E-6 (g/umol)
     |                 |___  'specificLeafArea' (g/cm2)=f{'time'} (day) x,y pairs :{ 0 0.001 10 0.002 40 0.003 80 0.003 }
     |___  'simulationControls'
           |___  'outputParameters'
                 |___  'table'
                       |___  'run' = 1
                       |___  'searchingDepth' = 10
                       |___  'startTime' = 0.
                       |___  'endTime' = 80.
                       |___  'timeInterval' = 0.5
```

User friendly editing of XML input files

Besides attaching a transformation sheet for transforming xml to html and view it in a webbrowser, a XML schema (xsd) is available, which allows schema aware editors to provide auto completion and validation of the input file. Below is a screenshot from an XML editor (plugin in www.eclipse.org) which shows the declaration of the schema, and a pop down menu for the available arguments for SimulaTable, and the different values that the objectGenerator argument can have.



```
<?xml version="1.0" encoding="UTF-8"?>
<SimulationModel
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="../scripts/XML/SimulaXMLSchema.xsd">
    <SimulaBase name="plants">
        <SimulaTable name_column1="time" > </SimulaTable>
    </SimulaBase>
</SimulationModel>
```

Attribute : objectGenerator
Data Type : objectGeneratorType
Enumerated Values :
- copyDefaults
- fieldPlantign
- generateRoot
- insertRootBranchContainers
- rootBranches
- rootBranchesOfTillers
- rootDataPoints
- seeding
- tillerDevelopment

...

```
endTime
function="getValuesFromPlantWaterUptake"
interpolationMethod="linear"
linksName
name
name_column2
objectGenerator="copyDefaults"
replacesPreviousDeclaration="true"
startTime
type="double"
unit
Press 'Ctrl+Space' to show XML Template Proposals
```

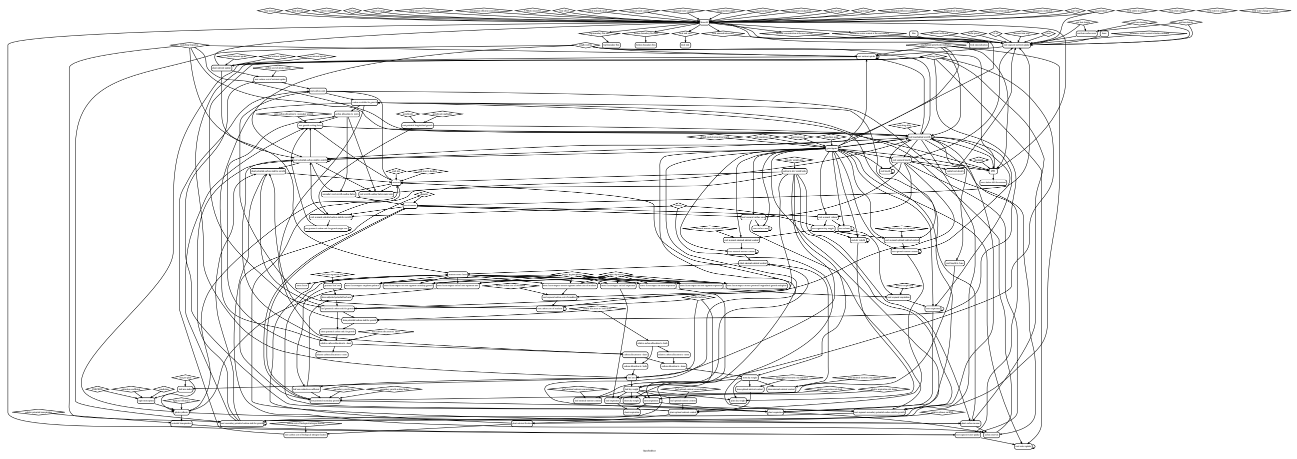
Property	
General	
endTime	
function	
interpolationMethod	
linksName	
name	
name_column1	time
name_column2	
objectGenerator	
replacesPreviousDeclaration	
startTime	
type	
unit	

Writable Smart Insert 6 : 42

Supplemental figure 6.1: Screenshot of XML editor in eclipse in which a new file was created, using the new file wizard. The schema is declared with "`xsi:noNamespaceSchemaLocation="../scripts/XML/SimulaXMLSchema.xsd"`" and the black and the black and white pop up boxes show suggestions, as defined in the schema.

Supplement 7: Diagram of all state variables and their dependencies in an exemplar bean simulation

We drew a graph which contains the various state variables in an example simulation and the dependencies among them. Each state variable is simulated by a SimulaObject, here we depicted SimulaConstants, SimulaTables and SimulaStochastic as wedges, whereas all others are depicted as a rounded boxes. The arrows indicate information flow, that is the result of one minimodel goes into the computation of another. The network is strongly dependent on the input file, and somewhat dependent on time, given that computations might switch on given conditions and should thereby be regarded as exemplar. To properly view the graph, enlarge the pdf strongly.



Supplemental figure 7.1: Graph representing all the state variables in a bean simulation, and their connections at day 12. For better viewing, enlarge by about 1200%.